Sicherheitsanalyse eines Mikroprozessors mit Busverschlüsselung

Diplomarbeit im Fach Informatik

vorgelegt von

Markus Kuhn

geboren am 1. Januar 1971 in München

Angefertigt am

Institut für Mathematische Maschinen und Datenverarbeitung (III)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: ***Prof. Dr. M. Dal Cin***
Dr. H. Hessenauer

Beginn der Arbeit: 01. 02. 1996
Abgabe der Arbeit: 31. 07. 1996

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Uttenreuth, den 31. 07. 1996

Markus Kuhn

Kurzfassung

Zahlreiche Sicherheitsanwendungen wie Kreditkartenterminals, Pay-TV-Verschlüsselungssysteme, elektronisches Bargeld und effektiver Softwarekopierschutz benötigen manipulationssichere Rechnerarchitekturen. Der unautorisierte Zugriff auf die in diesen Systemen ablaufende Software muss unmöglich sein, selbst wenn der Angreifer sehr gut ausgerüstet ist (Logik-Analysator, Chiptest-Labor, etc.) und über einen längeren Zeitraum hinweg unbeschränkten Zugang zur Hardware hat. Dazu werden seit einigen Jahren spezielle Krypto-Mikroprozessoren eingesetzt, bei denen der Daten- und Adressbus verschlüsselt genutzt wird. Die vorliegende Arbeit gibt einen Überblick über Konzepte für manipulationssichere Rechner und untersucht insbesondere einen heute im kommerziellen Bereich weit verbreiteten und bislang als sehr sicher geltenden Krypto-Prozessor: den DS5002FP Secure Microcontroller der Firma Dallas Semiconductor. Mit einem neuen Angriffsverfahren konnten dabei die Sicherheitsmechanismen dieses Prozessors mit überraschend einfachen Mitteln völlig umgangen werden. Diese neue Analysetechnik, welche die Entwickler des Prozessors und andere Sicherheitsstudien dazu völlig außer acht ließen, wurde im Rahmen der vorliegenden Diplomarbeit entwickelt, implementiert und unter realistischen Bedingungen erfolgreich demonstriert. Eine Reihe von neuen Schutzmaßnahmen gegen diesen Angriff kann die Sicherheit künftiger Prozessoren mit Busverschlüsselung erheblich verbessern. Darüber hinaus kann eine Reihe neuer Mechanismen den Einsatz derartiger Prozessoren unter modernen Betriebssystemen erlauben und dabei einen umfassenden Schutz der ausgeführten Software gegen Manipulation und Raubkopien bieten.

Abstract

Many computer security applications, like financial transaction terminals, pay-TV access control systems, electronic cash, and effective software protection, require tamper proof computer architectures. Unauthorized access to the software that is executed in these systems must be impossible, even if the attacker is very well equipped (logic analyzer, chip testing lab, etc.) and has full physical access to the hardware for an extended period of time. Over the past few years, special crypto-processors, which use an encrypted data and address bus, have been applied in such applications. This report provides an overview over various concepts for tamper proof computers. A security analysis is presented for the commercially widely used Dallas Semiconductor DS5002FP Secure Microcontroller, which has so far been considered to be highly secure. A new attack method allowed to defeat the security mechanisms of this processor completely using surprisingly simple technology. This new analysis technique, which has been completely ignored by the designers of this processor and in other security studies about it, has been developed, implemented, and demonstrated successfully under realistic conditions in this project. A number of new protection mechanisms against this attack can significantly increase the security of future bus encryption microprocessors. In addition, a range of new mechanisms will allow to use such processors with modern operating systems and will provide a comprehensive protection of the executed software against tampering and piracy.

Kuhn, M. *Sicherheitsanalyse eines Mikroprozessors mit Busverschlüsselung*. Diplomarbeit, Lehrstuhl für Rechnerstrukturen, IMMD III, Universität Erlangen-Nürnberg, Erlangen, Juli 1996.

Danksagung

Mein besonderer Dank gilt Jochen Stelzner für seine Unterstützung beim Aufbau der Versuchsanordnung, Uwe Hercksen für die Hilfe bei der Herstellung der Platine, Gabriele Roncolato von der Universität Mailand, Dr. Hansjürgen Hoeffgen und Peter Drescher vom Bundesamt für Sicherheit in der Informationstechnik, die mich zu dieser Arbeit angeregt haben, den Firmen Atlantik Elektronik und debis Systemhaus für die Bereitstellung von Literatur und kostenlosen Mustern des untersuchten Prozessors sowie meinen Betreuern für die Annahme dieses etwas aus dem Rahmen der normalen Lehrstuhlforschung fallenden Themas.

Inhaltsverzeichnis

1 Einleitung

In zahlreichen Computeranwendungen ist der Schutz des Systems vor böswilligen Manipulationen ein wichtiges Entwurfskriterium. Die Liste der möglichen Bedrohungen gegen die ein Datenverarbeitungssystem zu bestehen hat ist lang und vielfältig, und die folgende Aufzählung lässt das weite Einsatzfeld manipulationssicherer Rechner nur erahnen: Betrüger versuchen im Finanzbereich digitale Geldtransaktionen zum eigenen Vorteil umzulenken. Konkurrierende Unternehmen untersuchen Produkte und Entwicklungsunterlagen eines Herstellers um die eigene Entwicklungsarbeit zu vereinfachen. Softwareanwender versuchen die Lizenzbedingungen für Programme zu unterlaufen. Diebe versuchen gestohlene Hardware wieder in Betrieb zu nehmen. Geheimdienste, Privatdetektive und Konkurrenzunternehmen versuchen durch Manipulationen an Hardware und Software anderer an vertrauliche und wertvolle Informationen zu gelangen. Fernsehzuschauer versuchen abonnierbare verschlüsselte Pay-TV-Fernsehsender kostenlos zu empfangen und Fernsehhändler versuchen dazu geeignete billige Hardware anzubieten. Besitzer von im voraus bezahlbaren Elektrizitäts-, Gas- und Wasserzählern, wie sie häufig in Regionen der Dritten Welt eingesetzt werden, versuchen diese ohne Bezahlung der verbrauchten Ressourcen und ohne sichtbare Spuren freizuschalten. Terroristen versuchen große Unternehmen oder Behörden, deren Existenz vom reibungslosen Funktionieren der EDV abhängt, durch Manipulationen an deren Computersystemen zu schädigen. Kriegsführende Parteien versuchen durch die Untersuchung erbeuteter Hardware an Informationen zu gelangen, mit denen die Systeme des Gegners überwacht oder behindert werden können. Benutzer von digitalem Bargeld versuchen dieses zu vervielfältigen oder zu fälschen.

Im Folgenden werden wir all diese und andere an böswilligen Manipulationen von Computersystemen interessierten Personen oder Organisationen der Einfachheit halber unter der Bezeichnung „der Angreifer“ (engl. *attacker* oder *adversary*) zusammenfassen und die Manipulationsversuche selbst als „Angriff“ bezeichnen. Diese in der Literatur übliche Terminologie stammt zweifellos aus einer Zeit, als Computersicherheit und Kryptografie in erster Linie militärische Forschungsgebiete waren. Dieser Sprachgebrauch wird jedoch heute ebenso bei zivilen Sicherheitsanwendungen fortgeführt, die in letzter Zeit erheblich an Bedeutung gewonnen haben, und die heute den eigentlichen Schwerpunkt der Forschung und Entwicklung darstellen.

Die Forschung im Bereich der zivilen Computersicherheit konzentrierte sich lange Zeit vor allem auf Schutzmaßnahmen für zwei Situationen:

- **Sichere Betriebssysteme:** Teilen sich mehrere Benutzer den selben Rechner sowie die selben Massenspeicher und Kommunikationsmedien, so müssen Zugriffskontrollsysteme im Betriebssystem und in den Kommunikationsvermittlungseinrichtungen dafür sorgen, dass kein unbefugter Zugriff auf die Daten und Ressourcen der Mitbenutzer möglich ist. Die fundamentale Annahme beim Entwurf derartiger Zugriffskontrollsysteme ist, dass der Angreifer für seine Manipulationsversuche keinen direkten Zugang zur Systemhardware hat. Der Schutzmechanismus besteht aus der Verwaltung von Listen mit Zugriffsrechten, die von der Systemsoftware vor jedem Zugriff der Benutzer auf Ressourcen überprüft werden. Derartige Zugriffskontrollen lassen sich leicht umgehen, wenn der Angreifer das Betriebssystem selbst modifizieren kann oder sich direkten Zugang zur Hardware verschafft.

– **Sichere Datenübertragung:** Kommunizieren gefährdete Systeme über dem Angreifer leicht zugängliche Kommunikations- und Speichermedien wie Telefonleitungen, Internet, Funkverbindungen oder versandte Datenträger, so müssen mit kryptografischen Verfahren die ausgetauschten Daten sowohl gegen das Ausspähen (Vertraulichkeit der Daten) als auch gegen das Verändern (Integrität der Daten) durch Unbefugte gesichert werden [Sch96]. Auch hier wird wieder von der Annahme ausgegangen, dass der Angreifer nur Zugang zum Kommunikationsmedium hat, nicht aber zu den Endgeräten, in denen die Verschlüsselungs- und Authentisierungsalgorithmen implementiert sowie die dazugehörigen geheimen Schlüsseldaten gespeichert sind.

Im Vergleich zu den zahlreichen veröffentlichten Lösungsvorschlägen für die beiden genannten Aufgabenstellungen findet sich derzeit relativ wenig offene Literatur zu der Frage, wie Systeme zu entwickeln sind, bei denen der Angreifer auch vollen Zugang zur Hardware haben darf, in der die sicherheitsrelevanten Algorithmen ausgeführt werden. Im klassischen Rechenzentrum stellt sich diese Frage weniger, da nur eine kleine und sorgfältig ausgewählte Gruppe von Personen Zugang zu den Rechnern hat, und Angreifer durch Mauern und Personenzugangskontrollen abgehalten werden können.

1.1 Anwendungsbereiche für manipulationssichere Architekturen

Computertechnik wird schon lange nicht mehr nur in geschützten Rechenzentren und Gebäuden mit Zugangskontrolle eingesetzt. Die Regel sind heute verteilte Systeme, die aus einer großen Zahl von Endgeräten bestehen. Diese arbeiten oft unbeaufsichtigt wie etwa Geldautomaten und Telefonzellen, und sie werden in vielen Anwendungen sogar dem potentiellen Angreifer direkt ausgehändigt, wie etwa bei Pay-TV-Decodern, Chipkarten, Mobilfunktelefonen.

Die Sicherheit jeder kryptografischen Anwendung basiert letztendlich auf der physikalischen Sicherung der geheimen Schlüsseldaten gegen den Zugriff durch Unbefugte. Da verteilte Computersicherheits-Mechanismen in der Regel mit kryptografischen Techniken (z.B. Verschlüsselungs- und Unterschriftsalgorithmen) realisiert werden, ist die Aufgabe eines manipulationssicheren Systems in der Regel, geheime Schlüsseldaten zu schützen. Neben den eigentlichen Schlüsseldaten darf für den Angreifer natürlich auch die Ausführung der kryptografischen Algorithmen nicht beobachtbar sein, da aus dem Ablauf des Algorithmus die Schlüsseldaten offensichtlich wären. Neben diesen primär zu schützenden Daten und Algorithmen kann ein manipulationssicheres System noch weitere Anwendungssoftware und andere Komponenten enthalten, die für die Sicherheit des Gesamtsystems relevant sind.

Die ersten praktischen Überlegungen zum Schutz von geheimen Schlüsseldaten stammen, wie nicht anders zu erwarten, aus dem militärischen Bereich [Kah67]. So wurden beispielsweise auf Kriegsschiffen die Codebücher zur Chiffrierung geheimer Funkmeldungen mit Gewichten versehen und mit wasserlöslicher Farbe gedruckt, so dass die streng geheimen Informationen bei einem Überfall durch einfaches Überbordwerfen der Bücher schnell und zuverlässig vernichtet werden konnten. Andere Codebücher wurden auf Papier aus Nitrocellulose (Schießbaumwolle) gedruckt, damit sie im Notfall sekundenschnell rückstandsfrei verbrannt werden konnten. Spätere Beispiele von manipulationssicheren Systemen aus dem militärischen Bereich sind die Sicherheitsmechanismen in Kernwaffen,

die nach einem Diebstahl durch Terroristen das spaltbare Material unbrauchbar machen müssen sowie Sensoren, die zur Überprüfung von Abrüstungs- und Kernwaffenteststopp-Verträgen in anderen Ländern installiert werden [And96b].

Wir müssen davon ausgehen, dass der Angreifer technisch gut ausgebildet ist und über Ausrüstung zur Untersuchung und Änderung von elektronischen Schaltungen und hochintegrierten Halbleiterbausteinen verfügt.

In [Abr91] werden die beim Entwurf eines Systems zu berücksichtigenden Angreifer entsprechend ihren Möglichkeiten in drei grobe Gefährlichkeitsklassen eingeteilt:

- **Klasse I: Clevere Außenstehende.** Sie sind oft sehr intelligent, haben aber nur beschränktes Wissen über den Aufbau des untersuchten Systems. Sie haben nur Zugang zu mittelmäßig aufwendiger Ausrüstung (z.B. Lötkolben, Mikroskop, einfache Chemikalien, mechanische Werkzeuge, PC, In-Circuit-Emulator und Logik-Analysator). Sie nutzen oft existierende Schwächen des Systems aus, anstatt neue zu schaffen. Beispiele sind Studenten, Hobbyelektroniker oder Privatdetektive.
- **Klasse II: Erfahrene Insider.** Sie haben eine gezielte technische Ausbildung und viel Erfahrung. Sie haben unterschiedlich gutes Wissen über die Bestandteile des untersuchten Systems, aber prinzipiell Zugang zum meisten davon. Oft haben sie auch Zugang zu anspruchsvoller Ausrüstung zur Untersuchung des Systems. Beispiele sind einzelne Mitarbeiter eines Systemherstellers oder -betreibers.
- **Klasse III: Zahlungskräftige Organisationen.** Sie sind in der Lage, Spezialistenteams mit Experten verwandter und sich ergänzender Ausbildung zusammenzustellen, die mit großen finanziellen Mitteln ausgestattet sind. Sie können eine detailierte Analyse der untersuchten Systems durchführen, anspruchsvolle Angriffe entwickeln und haben Zugang zu den aufwendigsten Untersuchungshilfsmitteln (Chiptestarbeitsplätze, Elektronenmikroskope, Plasmaätzanlagen, Röntgengeräte, Elektronenstrahltester, Ionenstrahlarbeitsplatz, UV Lasersystem, uvam.). Sie können eventuell auf erfahrene Insider als Teammitglieder zurückgreifen. Beispiele für Klasse-III-Angreifer sind die Labors von Geheimdiensten, von großen Mikroelektronikherstellern und von mächtigen kriminellen Vereinigungen.

Im Folgenden betrachten wir kurz eine Reihe von Beispielanwendungen, die auf die Existenz einer manipulationssicheren Rechnerarchitektur angewiesen sind.

1.1.1 Verschlüsselte Pay-TV-Fernsehkanäle

In einigen Regionen der Welt ist inzwischen der Markt für private Fernsehsender, die sich ausschließlich über Werbeeinnahmen finanzieren können, gesättigt. Daher entstehen Sender, die von ihren Zuschauern Gebühren verlangen und dafür ohne Werbeunterbrechung viele teure Filme zeigen. Damit nicht auch andere Fernsehgerätebesitzer kostenlos in den Genuss des Pay-TV-Programms kommen, wird das Bild- und oft auch das Ton-Signal verzerrt. Nur zahlende Kunden erhalten ein Decoder-Zusatzgerät, das den ungestörten Empfang erlaubt.

Die erste Decodergeneration enthielt nur eine einfache analoge Verzerrungsschaltung, die meist die Bildsynchronsignale umcodierte. Da diese Geräte nicht auf kryptografischen

Algorithmen basierten, ließen sie sich sehr leicht nachbauen. Heute arbeiten Pay-TV-Decoder fast alle mit einem Verzerrverfahren, bei dem das analoge Fernsehbild kurz digital zwischengespeichert wird [New90, CEN92]. Dabei werden je nach System entweder die Bildzeilen untereinander vertauscht, oder es werden innerhalb jeder Bildzeile die Bildpunkte zyklisch verschoben. Der genaue Ablauf der Verzerrung (*scrambling*) wird durch einen Pseudozufallszahlengenerator bestimmt, der alle paar Sekunden auf einen neuen Startwert gesetzt wird. Der passende Startwert wird kryptografisch verschlüsselt in der Austastlücke des Fernsehbilds digital übertragen und zuerst entschlüsselt, bevor er an die Entzerrschaltung weitergegeben wird.

Die Entschlüsselung des Startwertes findet in einer Chipkarte statt, die in den Decoder gesteckt wird und die nur zahlenden Kunden ausgehändigt wird. Die Datenübertragungsrate in der Bildaustastlücke für die verschlüsselten Startwerte ist nicht sehr groß und der Startwert muss mindestens alle paar Sekunden neu übertragen werden, um eine schnelle Synchronisation des Decoders nach einem Kanalwechsel zu gewährleisten. Da der Startwert nicht für jede der vielen Millionen Kundenchipkarten getrennt verschlüsselt gesendet werden kann, erhält jede Kundenkarte den gleichen verschlüsselten Wert und muss daraus das gleiche Entschlüsselungsergebnis erzeugen wie alle anderen Karten. Jede Kundenkarte enthält daher die gleiche Entschlüsselungsfunktion mit den gleichen geheimen Schlüsseldaten.

Ein Angreifer wird nun versuchen, diese in allen Chipkarten gleiche Verschlüsselungsfunktion durch Analyse einer einzigen Chipkarte zu ermitteln. Gelingt ihm dies, so kann er sehr billig die Chipkarte duplizieren und mit hohem finanziellen Gewinn verkaufen. In diesem Fall ist der Sender gezwungen, mit großem finanziellen Aufwand die vielen Millionen Chipkarten der Kunden auszutauschen, um einen neuen noch unbekannten Schlüssel einsetzen zu können. Da der Verkauf von Piraten-Decoderkarten in der Vergangenheit zu größeren finanziellen Gewinnen für einige Angreifer führte, müsste der Schlüssel und der dazugehörige Entschlüsselungsalgorithmus so in der Chipkarte oder in einem anderen manipulationssicheren Modul untergebracht sein, dass ihn auch ein Klasse-II-Angreifer nicht ermitteln kann.

1.1.2 Mobiltelefonsonderangebote

GSM-Mobilfunktelefone werden gelegentlich zu einem sehr niedrigen Sonderpreis verkauft, jedoch mit einer Softwaresperre, die für beispielsweise ein Jahr nur den Gebrauch einer speziellen teureren Kundenkarte erlaubt. Der Angreifer wird versuchen, diese Telefone zum Sonderpreis zu kaufen und die Softwaresperre des Händlers rückgängig zu machen, so dass trotz des sehr niedrigen Kaufpreises jede Kundenkarte benutzt werden kann. Dann könnte der Angreifer mit großem Gewinn das Mobiltelefon zum normalen Preis weiterverkaufen. Daher muss der Hersteller dafür sorgen, dass die Software im Telefon gegen Manipulation gut geschützt ist und die Kartensperre zumindest von einem Klasse-I-Angreifer nicht gelöscht werden kann, ohne die Funktionsfähigkeit des Telefons zu beeinträchtigen.

1.1.3 Vorausbezahlte Kostenzähler

In einigen Gegenden, wie etwa in Südafrika, werden Stromzähler in Privathaushalten nicht regelmäßig zum in Rechnung stellen der verbrauchten Energie abgelesen. Statt dessen enthält jeder Zähler einen Mikroprozessor der ein Stromkonto verwaltet [And96a].

Der Strom wird über ein Relais abgeschaltet, sobald der Stromkontostand im Zähler auf Null steht. An Verkaufsstellen kann der Kunde einen Stromgutschein (*token*) über x kWh erwerben, mit dem er den Kontostand seines Zählers um x kWh erhöhen kann. Dieser Gutschein ist je nach System ein Datenpaket auf einer Magnetkarte oder es ist ein Papierstreifen mit einer 20-stelligen Dezimalzahl, die per Tastatur in den Zähler eingegeben werden muss. Vorausbezahlte Stromzähler haben sich insbesondere in den unterentwickelten *townships* Südafrikas bewährt, wo für die planlos errichteten Lehm- und Holzhütten der meisten Stromkunden weder ein Hausnummernsystem noch ein Postzustelldienst existiert und damit die Zählerablesung und Rechnungszustellung sehr problematisch wäre.

Jeder Zähler hat eine dem Kunden bekannte Seriennummer ID sowie einen im Mikroprozessor des Zählers geschützt abgespeicherten geheimen Schlüssel $K_{ID} = E_{K_V}(ID)$, der bei der Herstellung durch Verschlüsseln der Seriennummer ID mit einem Verkaufsschlüssel K_V durch einen Verschlüsselungsalgorithmus E berechnet wurde. An den Verkaufsstellen für Stromgutscheine sind Verkaufssysteme aufgestellt, die diese Gutscheine erzeugen. Der Schlüssel K_V wird im Verkaufssystem in einem Sicherheitsmodul mit einem manipulationsgesicherten Mikroprozessor gespeichert. Um einen Stromgutschein zu erwerben nennt der Kunde dem lokalen Verkäufer seine Zählernummer ID. Diese wird in das Verkaufssystem eingegeben, das daraufhin einen Stromgutschein G über die gewünschte Energiemenge erstellt und mit K_{ID} verschlüsselt, so dass der Gutschein $E_{K_{ID}}(G)$ nur an einem einzigen Zähler benutzt werden kann. Das Verkaufssystem notiert die verkaufte Strommenge im Sicherheitsprozessor und erstellt monatlich einen mit einem Berichtsschlüssel K_B verschlüsselten Verkaufsbericht, den der Verkäufer an die Elektrizitätsgesellschaft schicken muss, die ihm dann den verkauften Strom in Rechnung stellt.

Sollte es einem Besitzer oder Dieb eines Verkaufssystems gelingen, an die gespeicherten Daten im Sicherheitsmodul des Verkaufssystems zu gelangen, so könnte er den Schlüssel K_V in Erfahrung bringen, mit dem er kostenlos beliebig viele Stromgutscheine herstellen könnte. Sollte der Berichtsschlüssel K_B bekannt werden, so könnten Stromverkäufer alternativ auch die Verkaufsberichte für das Energieversorgungsunternehmen fälschen.

Die Sicherheitsanforderungen an den Prozessor eines Stromzählers sind nicht besonders gravierend, da sich hier Manipulationsversuche eher darauf konzentrieren, das Relais oder den ganzen Zähler zu überbrücken, was Spuren an Gehäuse und Kabelverplombung nach sich zieht. Wenn jedoch jemand die Software eines Verkaufssystems auslesen kann, so kann er damit Stromgutscheine für etwa 10 000 Kundenzähler fälschen. Daher kann das Energieversorgungsunternehmen dem lokalen Verkäufer und der Sicherheit in dessen Räumlichkeiten nicht vertrauen und darf ihm die geheimen Schlüssel K_V und K_B zur Erzeugung der Gutscheine und Berichte nur in einem manipulationssicheren Rechner im Verkaufssystem anvertrauen. Vergleichbare Konzepte werden auch andernorts eingesetzt, zum Beispiel in Großbritannien zur Portoabrechnung bei Brieffankiermaschinen.

1.1.4 Diebstahlsicherungen

In viele Mikroprozessorsysteme lässt sich sehr einfach eine Diebstahlsicherung einbauen, welche die Wiederinbetriebnahme von der Eingabe eines Geheimcodes abhängig macht. Moderne Autoradios verlangen beispielsweise eine Geheimzahl nach einem Ausfall der normalerweise ständig anliegenden Batteriespannung. Arbeitsplatzrechner lassen sich so konfigurieren, dass nach einem Neustart ein Passwort eingegeben werden muss. Damit der Dieb den Schutz nicht durch Anschließen einer Zusatzbatterie während des Abtransports

unterlaufen kann, könnten die Mikroprozessorsysteme auch regelmäßig eine Authentisierung mit einem anderen System über ein Datennetz durchführen, zum Beispiel im Fall des Autoradios mit dem Prozessor der Einspritzregelung oder einem Prozessor im Zündschlüssel, oder im Fall eines Arbeitsplatzrechners mit einem Dateiserver.

Auch diese Sicherheitskonzepte sind darauf angewiesen, dass der Dieb nicht einfach den Reaktivierungscode aus dem nichtflüchtigen Speicher des Rechners auslesen oder mit einem bekannten Code überschreiben kann, wie dies in der Regel bei heutigen Arbeitsplatzrechnern und Autoradios relativ einfach der Fall ist. Wenn der Reaktivierungscode zusammen mit der gesamten Steuersoftware in einem manipulationssicheren Modul untergebracht ist, so wird ein unbefugter Zugriff auf den Code nicht nur diesen, sondern auch die restliche Steuersoftware löschen, ohne die das gestohlene System funktionslos ist. Da die Steuersoftware auch nicht aus anderen Systemen gleichen Typs ausgelesen werden kann, selbst wenn deren Zugangscode bekannt ist, kann ein Dieb die gelöschte Steuersoftware auch nicht einfach wieder einspielen. Er müsste sie komplett neu entwickeln, was in der Regel den Diebstahl nicht mehr lohnenswert macht.

1.1.5 Effektiver Kopierschutz von Software

Software unterscheidet sich von allen anderen Produktarten dadurch, dass die Entwicklung sehr teuer ist, die Herstellung dagegen ohne Qualitätsverlust in beliebiger Stückzahl und praktisch kostenlos durch jeden Besitzer erfolgen kann. Dies führte zwangsläufig zur weiten Verbreitung von Raubkopien und entsprechend reduzierten Verkaufszahlen, einem gravierenden Problem für die weitere Entwicklung der Softwareindustrie.

Bislang ist es üblich, dass Softwarehersteller ihren Kunden den unverschlüsselten Maschinencode verkaufen, der in dieser Form auf jedem Computer gleichen Typs ablauffähig ist. Bei preiswerter Software für den Massenmarkt nimmt der Hersteller dabei in Kauf, dass in gewissem Umfang Raubkopien seines Produktes genutzt werden. Er vertraut darauf, dass die Illegalität der Nutzung von Raubkopien und die Gefahren von „Viren“ und „trojanischen Pferden“ beim Einsatz von Software aus dubiosen Quellen hinreichend viele Benutzer, insbesondere Firmen, dazu bringt, doch das Originalprodukt selbst zu erwerben. Nur bei sehr teurer Software für kleine Spezialmärkte (CAD, Entwicklungswerkzeuge, Expertensysteme, usw.) sowie bei Computerspielen haben sich einfache Kopierschutzmaßnahmen durchgesetzt. In der Regel wird entweder eine kleine Zusatzhardware (*dongle*) oder eine rechnerspezifische Seriennummer an vielen Stellen des Programms getestet. Ist die Zusatzhardware oder die zur Benutzerlizenz passende Seriennummer nicht vorhanden, so terminiert das Programm. Bei Computerspielen für den Massenmarkt sind diese Verfahren wenig praktikabel, und es haben sich andere durchgesetzt. Eine davon ist das Aufbringen von speziellen Sektoren auf die Diskette, die das Betriebssystem selbst so nicht erzeugen könnte und deren Vorhandensein unter Umgehung des Betriebssystems durch direkten Zugriff auf die Laufwerkssteuerung von der geschützten Software abgefragt wird. Dies verhindert zumindest die Vervielfältigung des Programms mit den normalen Dateikopierkommandos, verhindert aber auch eine legitime Sicherungskopie. Eine andere Vorgehensweise lässt den Benutzer vor Programmstart den Besitz der in kontrastarmen und daher schwer kopierbaren Farben gedruckten Programmanleitung beweisen, indem er einige vom Programm zufällig ausgewählte Stellen daraus zitieren muss.

All diese Ansätze bieten jedoch nur einen schwachen und vorübergehenden Schutz. Der ausführbare Maschinencode liegt im Klartext vor und der Angreifer kann diesen relativ

leicht disassemblieren, untersuchen und alle Sicherheitsabfragen die den Kopierschutz betreffen entfernen. Ein wirklich effektiver und dauerhafter Kopierschutz muss verhindern, dass der Angreifer Zugang zu den ausgeführten Maschinenbefehlen hat [Bes80, Ken81, Gol86, Whi87, Ost89]. Die Software wird dabei verschlüsselt ausgeliefert und existiert in unverschlüsselter Form außer beim Hersteller nur innerhalb der manipulationssicheren Rechner der Kunden. Ein geeignetes Schlüsselverwaltungsverfahren muss sicherstellen, dass nur Rechner, für die eine Softwarelizenz legal erworben wurde, den Schlüssel kennen, mit dem die ausgelieferte Software innerhalb des manipulationssicheren Bereichs vor der Ausführung entschlüsselt werden kann. Dieses Sicherheitsmodul ist wiederum so zu gestalten, dass alle Versuche es zu öffnen zur sofortigen Vernichtung der Schlüsseldaten führen. Ohne diese Schlüsseldaten ist die auf den Massenspeichern nur verschlüsselt abgelegte Software wertlos.

1.1.6 Elektronisches Bargeld

Derzeit gebräuchliche elektronische bargeld- und unterschriftslose Zahlungstechniken wie Euroscheckkarten oder Kreditkarten sind mit dem gravierenden Nachteil verbunden, dass der Händler für die Überprüfung des Zahlungsvorgangs eine teure Datenverbindung mit einem Bankcomputer benötigt. Nur so kann noch vor Übergabe der Ware im Laden sichergestellt werden, dass die Karte des Kunden nicht wegen Diebstahl gesperrt ist oder dass das Konto nicht überzogen wurde. Darüber hinaus bestehen bei Kreditkarten große Datenschutz-Bedenken, da der Kartenanbieter umfangreiches Datenmaterial über das genaue Konsumverhalten jedes Kunden gewinnt, da jede einzelne Ausgabe über ein Kundenkonto abgebucht wird.

Derzeit laufen verschiedene Pilotprojekte mit sogenannten „elektronischen Geldbörsen“. Dabei handelt es sich um kleine manipulationssichere Rechner, heute in der Regel Chipkarten, in denen ein Prozessor einen simulierten Bargeldbestand verwaltet. Über kryptografische Transaktionen kann der Kunde am Geldautomaten Bargeld auf seine Karte und dann im Laden von seiner Kundenkarte auf die Händlerkarte übertragen. Der Händler kann schließlich das in seiner Karte angefallene Geld über eine gelegentliche Telefonverbindung oder den Gang zum Geldautomaten auf sein Konto übertragen. Manche *electronic cash* Systeme unterscheiden garnicht zwischen Kunden- und Händlerkarte und erlauben Geldübertragungen in allen Richtungen zwischen allen beteiligten Karten.

Sollten die in den Karten eingesetzten Prozessoren nicht manipulationssicher sein, so bedeutet dies in einigen Systemen, dass der Angreifer beliebige Mengen von digitalem Bargeld zu seinen Gunsten erzeugen kann. Auf Grund der großen möglichen finanziellen Gewinne eines erfolgreichen Angreifers sollten die Entwickler einer elektronischen Geldbörse von einem Klasse-III-Angreifer ausgehen.

Ein vorsichtiges Systemdesign zeigt die für die Eurocheque-Chipkarte geplante Implementation einer elektronischen Geldbörse [ZAK95]. Hier wird für jede Kundenkarte ein Schattenkonto von der Bank geführt, auf dem der Betrag gespeichert wird, der sich derzeit maximal auf der Kundenkarte befinden kann. Dazu müssen in einem Gerät beim Händler die Transaktionen mit den Kundenkarten einzeln mitprotokolliert werden und diese Protokolle, die alle von der Händlerkarte unterschrieben werden, müssen schließlich an die Bankzentrale übermittelt werden. Diese kann dann bei jeder gemeldeten Zahlung eines Kunden den entsprechenden Betrag von Schattenkonto abbuchen. Nun taucht zwar wieder das Datenschutzproblem auf, die Bank erhält aber dafür ein Alarmsignal wenn eines der

Schattenkonten plötzlich einen negativen Betrag enthält. Dann hätte ein Angreifer seine Geldbörse durch geschickte Manipulation selbst aufgefüllt und hätte damit digitales Falschgeld in das System eingespeist.

Wenn diese Manipulation bei der Transaktion zwischen Kunden- und Händlerkarte erfolgt, so fällt der Betrug erst später bei der Abrechnung mit der Bank auf. Der Angreifer könnte mit einer manipulierten Kundenkarte bezahlen, die irgendeine beliebige Kundenkontonummer angibt und wäre damit nicht mehr im nachhinein identifizierbar. Die Kommunikation zwischen Händler und Kundenkarte wird über einen kundenkartenspezifischen Schlüssel gesichert, der in der Kundenkarte gespeichert ist und den die Händlerkarte mit einem Hauptschlüssel aus der Kundenkartennummer errechnen kann. Für die genauen Protokollabläufe des Bezahlvorganges sei hier auf [ZAK95] verwiesen. Sollte es einem Angreifer gelingen, die geheimen Hauptschlüsseldaten aus Händlerkarten auszulesen, so könnte er in diesem System Software für eine manipulierte Kundenkarte mit erfundener Kartennummer und passendem Kartenschlüssel entwickeln, mit der er beliebig oft ohne weitere Kosten bezahlen kann. Diese Gefahr ließe sich zwar prinzipiell durch asymmetrische Kryptosysteme abschwächen, worauf die Entwickler aber angesichts der für derzeitige Chipkartenprozessoren zu hohen Rechenleistungsanforderungen verzichtet haben. Die Sicherheit der derzeit geplanten elektronischen Geldbörsen hängt daher von der Manipulationssicherheit der verwendeten Chipkarten ab.

1.1.7 Beweissicherungssysteme

Manipulationssichere Module finden auch in Geräten Einsatz, die gewisse physikalische Bedingungen überwachen sollen. Ein Beispiel dafür sind Sensoren, die es verschiedenen Nationen erlauben, gegenseitig die Einhaltung von Kernwaffenteststopp-Verträgen aus der Ferne zu überwachen. Dabei handelt es sich um Seismometer, die ständig verschlüsselt die aufgezeichneten Erschütterungen an einen Satelliten senden. Sobald Unbefugte den Sensor öffnen wird sofort der kryptografische Schlüssel mit dem die gesendeten Daten unterschrieben werden gelöscht, so dass auch nach einer Manipulation am Sensor keine korrekt unterschriebenen Daten mehr gesendet werden können und der Manipulationsversuch der Überwachungsstelle auffällt [And96b]. Ähnliche Mechanismen finden sich in Einbruchmeldeanlagen und Kraftfahrzeug-Fahrtenschreibern.

1.2 Konzepte für manipulationssichere Systeme

Der Aufwand der Schutzmechanismen, die in einem manipulationssicheren System eingesetzt werden müssen, hängt von den geschätzten Fähigkeiten und Mitteln des Angreifers ab. In [Abr91] wird die Klassifikation von Schutzmechanismen in die folgenden sechs Stufen vorgeschlagen, abhängig von den Kosten die aufgewendet werden müssen, um den Sicherheitsmechanismus zu überlisten:

- **ZERO:** Keine speziellen Sicherheitsvorkehrungen. Beispiel: ein normaler IBM PC in einem frei zugänglichen Raum.
- **LOW:** Es existieren einige einfache Sicherheitsmechanismen, die sich jedoch mit üblichen Laborhilfsmitteln und Werkzeugen wie Kneifzange, Lötkolben, Mikroskop, usw. umgehen lassen.

- **MODL:** Teurere Werkzeuge sowie gewisses spezielles Wissen werden für einen erfolgreichen Angriff benötigt. Die Werkzeugkosten können im Bereich von etwa 500 bis 5000 US-Dollar liegen.
- **MOD:** Spezielle Ausrüstung sowie spezielles Können und Wissen werden für einen erfolgreichen Angriff benötigt. Werkzeuge und Ausrüstung können im Bereich 50 000 bis 200 000 US-Dollar kosten. Der Angriff kann zeitaufwendig, aber letztendlich doch erfolgreich sein.
- **MODH:** Die erforderliche Ausrüstung ist zwar verfügbar, aber sehr teuer in der Anschaffung und im Betrieb und kann zwischen 50 000 bis 200 000 oder mehr US-Dollar kosten. Spezielles Können und Wissen ist notwendig um die Ausrüstung einsetzen zu können. Mehr als ein aufwendiger Vorgang kann zur Überwindung der Sicherheitsmechanismen notwendig sein, so dass mehrere Experten mit sich ergänzendem Wissen zusammenarbeiten müssen. Der Angriff könnte letztendlich erfolglos bleiben.
- **HIGH:** Alle bekannten Angriffsversuche waren erfolglos. Eine Untersuchung durch ein Team von Spezialisten ist erforderlich. Sehr spezielle Ausrüstung ist erforderlich, die zum Teil erst entworfen und hergestellt werden muss. Die Gesamtkosten des Angriffs können mehrere Millionen US-Dollar übersteigen und es ist unsicher ob der Angriff erfolgreich sein wird.

Diese Klassifikation von Sicherheitsmechanismen ist sicher sehr wertvoll für den Anwender eines Sicherheitsmoduls, da eine handfeste Aussage über den für einen Angriff notwendigen Aufwand getroffen wird. Organisationen welche Sicherheitsprodukte zertifizieren vermeiden jedoch gerne konkrete Aussagen über die minimal erforderlichen Kosten eines erfolgreichen Angriffs, da eine gute Idee, eine sehr versteckte Sicherheitslücke oder eine neu verfügbare Technologie ganz unerwartet erheblich die Kosten des günstigsten möglichen Angriffs reduzieren kann.

Für Zertifizierungszwecke wurde daher in einer entsprechenden US-Norm [FIP94] eine alternative Grobklassifikation von Sicherheitsmaßnahmen für manipulationssichere kryptografische Module vorgenommen. In den vier FIPS-Sicherheitslevels werden nur grundlegende Anforderungen an die Aufgaben der implementierten Schutzmechanismen gestellt, jedoch keine Aussagen über die Kosten eines Angriffs gemacht:

- **Sicherheitslevel 1:** Es werden nur Anforderungen an die verwendeten kryptografischen Algorithmen gestellt. Es werden keine besonderen physikalischen Schutzmaßnahmen verlangt, die über die normalen bei elektronischen Geräten üblichen geschlossenen Gehäuseformen hinausgehen.
- **Sicherheitslevel 2:** Das Sicherheitsmodul muss mit einem Siegel oder einem Schloss gesichert sein, oder in ein undurchsichtiges Material vergossen sein, so dass ein einfacher Manipulationsversuch durch die dabei entstandene Beschädigungen im Nachhinein offensichtlich wird.
- **Sicherheitslevel 3:** Ein Manipulationsversuch soll nicht nur im Nachhinein als Beschädigung erkennbar sein, sondern bereits während des Eingriffs vom Modul erkannt werden und zur sofortigen Vernichtung der im Sicherheitsmodul gespeicherten Geheimnisse führen. Ein einfaches Beispiel wäre ein sehr stabiles Gehäuse mit Schaltern, die beim Abnehmen des Gehäusedeckels sofort die gespeicherten Schlüsseldaten löschen, oder das Vergießen der Schaltung in sehr hartem Epoxid-Harz, um eine Beschädigung der Schaltung bei der Untersuchung wahrscheinlicher zu machen.

– **Sicherheitslevel 4:** Mit gewissem mechanischem Aufwand kann es immer noch relativ leicht möglich sein, die Alarmmechanismen von Level 3 Modulen zu umgehen (zum Beispiel indem dünne Löcher an den Sensoren vorbei durch das Gehäuse gebohrt werden über die der Zugang erfolgt). Level 4 verlangt einen umfassenden Eindringschutz von allen Seiten her. Das Eindringsensorsystem muss das Modul wie eine lückenlose Schutzhülle umgeben und sofort im Falle eines Angriffs die Geheimdaten löschen. Darüber hinaus muss durch Tests sichergestellt werden, dass bei variablen Umwelteinflüssen wie etwa Temperatur- und Spannungsschwankungen keine die Sicherheit des Moduls gefährdenden Systemzustände eintreten können. Alternativ können Sensoren eingesetzt werden, die bei ungewöhnlichen Umweltbedingungen wie etwa extremer Kälte eine Löschung der Geheimdaten auslösen.

Bislang werden in kommerziellen Anwendungen im Wesentlichen drei verschiedene Ansätze zum Entwurf manipulationssicherer Rechner eingesetzt, die im Folgenden näher erläutert sind.

1.2.1 Einchip-Systeme

Sofern die zu schützende Software nur wenige tausend Bytes lang ist, kann sie in nichtflüchtigem Speicher zusammen mit der CPU in einem einzigen Microcontroller-Chip untergebracht werden. Masken-ROM ist die kompakteste Speicherform auf einem Chip, aber da die Information durch das Chiplayout festgelegt ist, lässt sich ein ROM mit einem Elektronenmikroskop relativ problemlos auslesen (zumindest solange die ROM-Bits sich nicht nur durch die Dotierungsmuster unterscheiden). Zudem lassen sich Schlüsseldaten im ROM nicht nachträglich ändern oder löschen. Als nicht-flüchtige Speicher in Einchip-Systemen haben sich vor allem EEPROMs durchgesetzt. Dabei handelt es sich um Feldeffekttransistoren, deren Gate-Eingang völlig von isolierendem Material umgeben ist. Durch eine hohe Programmierspannung kann diesem *floating gate* ein bestimmtes Potential aufgezwungen werden, das dann den Schaltzustand des Transistors über viele Jahre hinweg bestimmt, womit ein Bit abgespeichert wird.

Eine sehr weit verbreitete Anwendungsform von EEPROM-Einchip-Systemen sind Chipkarten [ISO87]. Dabei wird der Microcontrollerchip auf ein etwa 1 cm^2 großes dünnes Kunststoffplättchen geklebt, das auf der gegenüberliegenden Seite über meist sechs oder acht Kontaktflächen verfügt. Der Siliziumchip wird mit sehr dünnen Gold- oder Aluminium-Bondingdrähten wie in einem normalen Chipgehäuse mit den Kontaktflächen elektrisch verbunden und anschließend in Epoxid-Harz vergossen. Dieses Chipmodul wird zur besseren Handhabbarkeit in eine größere Kunststoffform integriert, zum Beispiel im ISO-Kreditkartenformat, im Miniaturkartenformat wie es bei GSM Mobiltelefonen eingesetzt wird, oder in einer Plastikschlüsselform, wie sie bei einigen Pay-TV-Decodern und vorausbezahlten Elektrizitätszählern zu finden ist.

Über die externen Kontakte wird der Kartenprozessor mit 5 Volt Versorgungsspannung, einem Taktsignal mit meist etwa 3,5 MHz Frequenz sowie einem Resetsignal versorgt. Ältere Bausteine müssen daneben noch mit 12–25 Volt Programmierspannung versorgt werden, während neuere Kartenprozessoren diese über einen Oszillator und einen Dioden-Kondensator-Spannungsvervielfacher selbst auf dem Chip aus der 5 V Versorgungsspannung herstellen können. Die Kommunikation mit der Außenwelt erfolgt mit einem einzigen I/O-Kontakt über ein asynchrones Halbduplex-Protokoll mit üblicherweise 9600 bis

38 400 bit/s Datenrate. Spezielle Chipkartenprozessoren sind heute mit bis zu 20 Kilobyte EEPROM-Speicher und bis zu einigen hundert Byte RAM verfügbar. Chipkarten im Kreditkartenformat müssen starken Biegebeanspruchungen standhalten, da sie oft in Brieftaschen aufbewahrt werden. Diese mechanischen Anforderungen beschränken die maximale Seitenlänge eines Prozessorchips auf etwa 5 mm. Daneben müssen Chipkartenprozessoren besonders gegen elektrostatische Entladungen durch Berühren der Kontakte und gegen die starken Magnetfelder in Magnetkartenschreibgeräten ausgelegt sein.

Die Sicherheit von Einchip-Systemen wird von den Herstellern dadurch begründet, dass dem Angreifer die Systembusleitungen nicht zugänglich sind und er daher nur über die serielle Schnittstelle mit der Anwendungssoftware kommunizieren kann. Ferner wird von den Herstellern darauf hingewiesen, dass in EEPROM-Speichern die geheime Software nicht optisch sichtbar ist und lediglich als sehr empfindliches Ladungsmuster aufbewahrt wird, das sich beim Abätzen der oberen Schutzschichten sofort verflüchtigen wird.

Dennoch lässt sich aus Chipkartenprozessoren mit gewissem Aufwand die geheime Software auslesen. Das Epoxid-Harz, in das der Chip eingebettet ist, kann mit rauchender Salpetersäure ($> 98\,\%$ HNO_3) aufgelöst und mit Aceton entfernt werden. Salpetersäure kann chemisch die in Siliziumchips eingesetzten Materialien Silizium, Siliziumoxid, Siliziumnitrit und Gold nicht angreifen. Das für Leiterbahnen auf den Chip aufgedampfte Aluminium überzieht sich sofort mit einer resistenten Schutzschicht (Al_2O_3) und wird daher ebenfalls nicht geschädigt, solange kein Wasser anwesend ist. Durch mehrere Ätzversuche mit Telefonkarten und mit Chipkarten aus Pay-TV-Systemen, die Microcontroller der Hersteller Siemens, Motorola, und SGS Thompson enthielten, konnte ich zeigen, dass durch das Entfernen des Epoxid-Harzes mit erhitzter rauchender Salpetersäure und Aceton sowie durch mehrstündige Lichteinwirkung weder der Inhalt des EEPROM-Speichers noch die Funktionstüchtigkeit des Prozessors beeinträchtigt wird.

Schon normale EPROM-Microcontroller, die nicht speziell für Chipkarten- oder Sicherheitsanwendungen ausgelegt sind, versuchen das unbefugte Auslesen der Software zu verhindern. Sie verfügen über eine spezielle EPROM-Speicherzelle mit einem Sicherheitsbit. Falls es gesetzt ist, wird das einfache Auslesen über die Programmierverifikationsfunktion des Prozessors verhindert. Jedoch ist bei den meisten Microcontrollern dieses Sicherheitsbit in einer EPROM-Zelle außerhalb der Fläche des normalen EPROM-Speichers untergebracht. Der Angreifer muss daher nur die Chipverpackung wie beschrieben entfernen, den EPROM-Speicher mit Farbe abdecken und die restliche Chipfläche mit UV-Licht bestrahlen, um das Sicherheitsbit zu löschen, ohne die Programmdaten zu vernichten. Anschließend kann die Software über den Programmiermodus ausgelesen werden. Diese Technik ist auch für Klasse-I-Angreifer durchführbar. Chipkartenprozessoren für Sicherheitsanwendungen verfügen in der Regel über bessere Schutzmechanismen.

Auch wenn es sehr schwierig ist, die Potentiale der EEPROM-Speicherzellen einer zugänglichen Chipoberfläche direkt auszulesen, so ist es doch relativ leicht möglich, Zugriff zu den Busleitungen zu bekommen. Nachdem die Epoxid-Harz-Verpackung des Chips entfernt wurde, befindet sich zwischen der Metallisierungsschicht, in der die Aluminiumverbindungen zwischen den Transistoren liegen, und der Chipoberfläche nur noch eine Passivierungsschicht. Diese robuste isolierende Schutzschicht aus Siliziumnitrit oder Siliziumoxid schützt die tieferen Schichten vor Beschädigung, Umwelteinflüssen und Ionenmigration. Sie lässt sich beispielsweise mit Hydrogenfluorid-Plasmaätzverfahren entfernen. Anschließend kann der Angreifer feine Microprobing-Metallnadeln unter einem stark vergrößernden

Mikroskop auf einem vibrationsgedämpften Arbeitstisch auf die ihn interessierenden Busleitungen setzen. Diese 0,5–2 μm spitzen Nadeln können über einen Vorverstärker mit einem Logik-Analysator verbunden werden, der dann die Vorgänge auf dem untersuchten Prozessorbus aufzeichnet.

Einige Sicherheitsprozessoren verfügen über Kapazitätssensoren, die das Entfernen der Passivierungsschicht erkennen können sollen. Jedoch wirken diese Sensoren nur lokal und sind aufgrund ihrer großen Fläche leicht zu erkennen. Vor dem Wegätzen der Passivierungsschicht können diese Schutzsensoren daher mit einem Schutzlack abgedeckt werden. Es sind auch Prüfnadeln verfügbar, die in der Halterung einen Ultraschallgenerator integriert haben, mit dem die Passivierungsschicht nur genau unterhalb der Nadelspitze entfernt wird.

Ein aufwendigeres Untersuchungsverfahren sind Elektronenstrahltester, bei denen der Chip wie in einem Elektronenrastermikroskop abgetastet wird. Die Anzahl und Energie der von den Primärelektroden des Kathodenstrahls aus der Chipoberfläche herausgeschlagenen Sekundärelektroden geben Auskunft über das lokale elektrische Potential. Auf dem Bildschirm des Elektronenstrahltesters sind daher die Spannungen auf den Leitungen des Chips als Helligkeitsunterschiede erkennbar (*voltage contrast*). Weitere aufwendigere Untersuchungsverfahren sind das Rastertunnelmikroskop, bei dem der Strom einer dicht über dem Chip schwebenden Metallnadel gemessen wird, und optische Verfahren, bei denen ein Kristall auf die Chipoberfläche gebracht wird, der seine optischen Eigenschaften abhängig von elektrischen Feldern ändert, die dann mit Laserstrahlen abgetastet werden können. Diese Verfahren können bei Integrationsdichten und Frequenzen eingesetzt werden, bei denen Microprobing-Nadeln unzureichend sind.

Durch Beobachten der Busleitungen des Prozessors kann der geheime Speicherinhalt mitprotokolliert werden. Alternativ kann auch der Prozessor angehalten werden, und über die Busleitungen wird aktiv vom Angreifer der Speicher ausgelesen. Anschließend wird der vorgefundene Maschinencode disassembliert und ausgewertet, womit der Angriff auf das Einchip-System erfolgreich war.

Eine denkbare Schutzmaßnahme wäre eine zusätzliche Metallisierungsschicht, die der Angreifer jedoch ebenfalls durch selektive Ätzverfahren entfernen kann. Seit wenigen Jahren stehen sogar Geräte zur Verfügung, die mit fokusierten Ionenstrahlen fast beliebige Manipulationen auf Chipoberflächen durchführen können. Ionenstrahlen können Substanzen entfernen und dadurch Leitungen unterbrechen oder von der umgebenden Isolation befreien. Sie können aber auch leitende und isolierende Schichten mit großer Präzision auf der Chipoberfläche ablagern, so dass sogar Durchkontaktierungen zu tieferen Schichten hergestellt und die vorhandenen Schutzschaltungen im Chip regelrecht umgebaut werden können. Mit diesen Werkzeugen ist sogar der Zugang zu in den unteren Chipschichten vergrabenen Busleitungen möglich.

Durch systematisches Abätzen dünner Schichten des Chips und anschließendem vollständigen Fotografieren der Oberfläche des Chips kann ein hochauflösendes dreidimensionales Modell des Chips erstellt werden, aus dem dann mit Bildverarbeitungstechniken die Netzliste der Schaltung rekonstruiert werden kann [Bly93]. Derartige den Chip zerstörende Verfahren helfen dem Angreifer, in Hardware implementierte kryptografische Algorithmen zu verstehen und einen Ausleseangriff auf einen anderen noch intakten Chip mit identischem Layout vorzubereiten.

Angesichts dieser heute in vielen besseren Mikroelektroniklabors verfügbaren Techniken

erscheint es ausgesprochen schwierig, wirkungsvolle Sicherheitsmechanismen auf Chipebene zu realisieren. Die meisten heutigen Chipkarten dürften daher nur die MOD Sicherheitsstufe auf der Skala nach [Abr91] erreichen. Einige Mikroelektronik-Labors bieten angeblich sogar kommerziell das Auslesen von Einchip-Systemen in Chipkarten und anderen Systemen als Dienstleistung für etwa zehntausend US-Dollar pro Chip an.

Für Einchip-Systeme stellt daher [FIP84] keine speziellen Anforderungen an Schutzmaßnahmen auf dem Chip selbst, sondern nur an die Verpackung des Chips. Im Sicherheitslevel 4 muss der Chip in ein hartes undurchsichtiges Material eingegossen werden, dessen Härte- und Adhäsionseigenschaften es sehr wahrscheinlich machen, dass bei einem Versuch die Verpackung mechanisch zu entfernen der Chip ernsthaft beschädigt wird. Die Löslichkeitseigenschaften des Vergussmaterials sind so mit den auf dem Chip verwendeten Substanzen abzustimmen, dass Versuche, das Verpackungsmaterial chemisch aufzulösen, auch den Chip angreifen werden [Pri86]. Derartige Sicherheitsverpackungen sind aber heute zumindest bei kommerziellen Sicherheits-Chips nicht gebräuchlich.

1.2.2 Sichere Verpackung ganzer Baugruppen

Das grundlegende Problem von EEPROM-basierten Sicherheitsmodulen besteht darin, dass externe Energie benötigt wird, um die geheimen Daten vernichten zu können. Sobald der Angreifer diese Energiezufuhr unterbricht sind die Daten gefangen und der Zugriff auf die Daten kann in aller Ruhe durchgeführt werden, ohne dass das Auslösen eines Alarmsystems befürchtet werden muss.

Eine attraktive Lösung besteht darin, die geheimen Daten in batteriegepuffertem statischem RAM unterzubringen. Da nun bereits eine Energiequelle erforderlich ist, um die Daten zu erhalten, kann die selbe Energiequelle auch zum Betrieb von wirkungsvollen Alarmmechanismen genutzt werden, die im Fall eines Eindringens in die Schutzhülle des Sicherheitsmoduls einfach die Stromversorgung des SRAM-Speichers unterbrechen. Der Angreifer kann nicht mehr ohne weiteres die Energiequelle entfernen, ohne dadurch die Daten zu gefährden. CMOS SRAM-Bausteine mit 128 Kilobyte Kapazität sind heute mit Erhaltungsströmen von unter 1 μA erhältlich, was eine kleine Lithiumbatterie problemlos mindestens ein Jahrzehnt zur Verfügung stellen kann.

In [Wei87] wurde der Entwurf eines SRAM-Sicherheitsmoduls bei IBM im Rahmen des μABYSS-Projekts dokumentiert, in dem eine ganze 8×20 cm große Baugruppe mit einem Prozessor, mehreren Speicher- und Peripheriechips, einer kleinen Alarmschaltung und einer Batterie untergebracht ist. Diese Baugruppe wird auf allen Seiten völlig lückenlos und in mehreren Lagen mit einem 80 μm dünnen isolierten Draht umwickelt. Anschließend wird die so umwickelte Baugruppe in ein hartes Epoxid-Harz eingegossen. In diesem Kunststoff befinden sich Aluminium- und Siliziumflocken. Sie sollen die maschinelle Bearbeitung der Verpackung oder deren Entfernung durch einen UV-Laser ohne Beschädigung der Drahtumwicklung oder deren Isolation wesentlich erschweren. Die Isolation des Wickeldrahtes ist chemisch weniger widerstandsfähig als das Vergussmaterial, so dass bei einem Aufätzversuch mit großer Wahrscheinlichkeit Kurzschlüsse entstehen werden. Eine Schaltung aus Operationsverstärkern vergleicht ständig die Widerstände der beiden langen Drahtschleifen, die um die Baugruppe gewunden wurden. Wenn sich durch eine Unterbrechung oder einen Kurzschluss die Widerstandsverhältnisse deutlich ändern, so wird sofort die Spannungsversorgung der SRAM-Chips mit Masse kurzgeschlossen, so dass die Daten gelöscht werden. Diese Schaltung benötigt nur etwa 20 μA Strom.

Bei derartigen Sicherheitsmodulen ist zu beachten, dass SRAM-Bausteine ihren Speicherinhalt bei Raumtemperatur für einige Sekunden ohne Versorgungsspannung erhalten können. Die verbliebene Ladungsverteilung in den Feldeffekttransistoren reicht aus, um das Flip-Flop beim Wiedereinschalten in den alten stabilen Zustand zurückkehren zu lassen. Diese Ladungsverteilung wird durch thermisches Rauschen langsam verändert. Durch Kühlung der ganzen Baugruppe mit flüssigem Stickstoff oder Helium kann das thermische Rauschen soweit verringert werden, dass sich die Information über eine Stunde ohne Versorgungsspannung halten kann.

Ein Angreifer könnte das Modul abkühlen und ohne Rücksicht auf den Alarmmechanismus die Verpackung schnell entfernen. Dann würde er den Kurzschluss, den der Alarmmechanismus ausgelöst hat, beseitigen und eine externe Versorgungsspannung anlegen, um anschließend das Modul auszulesen. Als Gegenmaßname sollte daher ein Sensor bei einer Abkühlung deutlich unter übliche Temperaturen ein Überschreiben der Daten auslösen.

Mit dieser Technik lassen sich Sicherheitsmodule entwickeln, die den FIPS Level 4 Anforderungen genügen, und auch einem Klasse-III-Angreifer einiges Kopfzerbrechen bereiten dürften. Ein möglicher Angriff wäre, bei Raumtemperatur mit speziell entwickelten Werkzeugen in Sekundenbruchteilen die Alarmmechanismen und die Batterie mechanisch abzutrennen um ein aktives Überschreiben der Daten zu verhindern, um dann sofort die Speicherbausteine mit einer externen Betriebsspannung versorgen. Ein anderer Angriffsansatz wäre, mechanisch die Drähte teilweise freizulegen, um mit einem sehr empfindlichen Spannungsmessgerät die Drahtwindungen nahe der Alarmschaltung ausfindig zu machen. Die an diesen Windungen anliegenden Potentiale werden dann mit einer externen Präzisionsspannungsquelle festgehalten, woraufhin der Angreifer die zwischen den extern spannungsstabilisierten Windungen liegenden anderen Drahtwindungen unterbrechen und den freigewordenen Raum für einen Zugang zum Inneren des Moduls nutzen kann.

Das physikalische Sicherungssystem des *IBM Transaction Security System* Moduls, das beispielsweise in vielen Geldautomaten eingesetzt wird, ist kurz in [Abr91] beschrieben. Die äußere Schutzhülle besteht aus einer flexiblen Membran, auf die mit elektrisch leitendem Lack ein Leiterbahnmuster aufgedruckt wurde, das die gesamte Fläche ausfüllt. Diese Membran umgibt die sicherheitsrelevanten Speicher-, Prozessor- und Verschlüsselungsbausteine. Sie ist mit einer Schutzschicht umgeben, die dem Membranmaterial chemisch sehr ähnlich ist. Die Widerstandsüberwachungsschaltung entspricht der in [Wei87] beschriebenen. Innerhalb der Schutzmembran befinden sich weitere Alarmmechanismen, unter anderem ein Temperatursensor. Laut [Abr91] erfüllt dieses Modul die MODH Anforderungen und bietet damit einen guten Schutz gegen Klasse-II-Angreifer. Ein Sicherheitsmodul für die HIGH-Anforderungen gegen Klasse-III-Angreifer hätte laut [Abr91] leicht die doppelten Kosten des Gesamtsystems verursacht.

1.2.3 Busverschlüsselungssysteme

Sichere Verpackungen für ganze Baugruppen können einen recht beeindruckenden Schutz zumindest gegen Klasse-II-Angreifer bieten, jedoch sind sie auch mit einer Reihe von Nachteilen verbunden:

– Je größer das Sicherheitsgehäuse ist, desto empfindlicher wird die Schutzhülle gegen normale Vibrationen, Kräfte und Temperaturschwankungen, was die Wahrscheinlichkeit von Fehlalarmen erhöht und damit die Zuverlässigkeit des Systems reduziert.

- Das Sicherheitsgehäuse muss hermetisch dicht sein, denn sonst könnte der Angreifer mit feinen Drähten Kontakt zu den Komponenten herstellen. Daher ist keine direkte Belüftung des Systems möglich, wodurch die Gesamtleistung die innerhalb der Schutzhülle verbraucht werden darf auf wenige Watt eingeschränkt wird.
- Es sind aufwendige Herstellungsverfahren notwendig, die speziell an die gewünschte Gehäuseform angepasst werden müssen. Dadurch entstehen hohe Kosten.
- Der Platzbedarf einer sicheren Verpackung kann bei kleinen portablen Geräten problematisch sein.

Diese Nachteile lassen sich vermeiden, wenn die Sicherheitshülle auf einen einzigen Chip reduziert und mit den Sicherheitsvorteilen der batteriegepufferten SRAM-Speicherung von Geheimdaten gekoppelt wird. Bei der Ausführung von geheimer Software mit geheimen Daten in einem manipulationssicheren System ist es prinzipiell ausreichend, wenn diese in entschlüsselter Form nur innerhalb des Prozessors vorliegen. Dazu wird eine Ver- und Entschlüsselungslogik geeignet auf den Chip eines speziellen Sicherheitsprozessors integriert, so dass auf den externen Bussen und den angeschlossenen normalen Speicherbausteinen die verarbeiteten Daten ausschließlich verschlüsselt vorliegen. Der Angreifer darf im Prinzip in so einem System vollen Zugang zu den externen Systembussen, den Speicherbausteinen und den Peripheriechips haben, ohne dass er daraus die geschützten verarbeiteten Daten rekonstruieren kann. Dieses Softwareschutzkonzept wurde erstmals von Robert M. Best in einem Patent 1979 beschrieben [Bes79, Bes80, Bes81, Bes84b].

Die einzigen Systembereiche, die bei einem Busverschlüsselungskonzept nicht vom Angreifer beobachtbar sein dürfen, sind der Prozessor, die Busverschlüsselungslogik sowie das Schlüsselregister das bestimmt, wie die verarbeitete Software chiffriert wurde. Angesichts der Möglichkeiten, EEPROM-basierte Einchip-Sicherheitsprozessoren zu überwinden, empfiehlt sich, die Schlüsseldaten nur in SRAM zu speichern. Der Busverschlüsselungs-Prozessor verfügt über einen eigenen Batterieanschluss und speichert die Schlüsseldaten in einem statischen batteriegepufferten Schlüsselregister ab. Der Prozessor enthält auf dem Chip verschiedene Sensoren, die bei Angriffsversuchen sofort das Schlüsselregister löschen, ohne dessen Inhalt die extern gespeicherte Software nutzlos ist.

Die bei Prozessoren mit EEPROM-Speichern so erfolgreichen Angriffsverfahren werden erheblich dadurch kompliziert, dass während des Zugriffs auf den Chip niemals die Spannungsversorgung für das Schlüsselregister unterbrochen werden darf. Chemikalien zum Entfernen der Sicherheitsverpackung und Passivierungsschicht sind oft leitfähig und können Kurzschlüsse auf dem Chip hervorrufen. Dank der für das Schlüsselregister vorhandenen Batteriespannung lassen sich leicht aktive Sensoren realisieren, die sehr empfindlich beispielsweise auf Licht, extreme Temperaturen, Chemikalien, Kapazitätsänderungen, oder das Entfernen der oberen Schutzschichten reagieren.

Gute Busverschlüsselungssysteme benötigen keine speziellen Fertigungsverfahren für die Baugruppe, da der Prozessorchip in einem herkömmlichen Gehäuse untergebracht ist und mit normalen Bestückungsmaschinen eingesetzt werden kann. Da der Hauptspeicher frei zugänglich sein darf, gibt es keine Probleme mit der Belüftung oder Wartung, und die Speicherkapazität ist nicht begrenzt. Dennoch lassen sich Busverschlüsselungssysteme besonders leicht mit einer sicheren Verpackung kombinieren, da auch ein externer Alarmmechanismus das Schlüsselregister löschen kann, wenn ein unabhängiger zweiter Schutzschirm zur Erhöhung der Gesamtsicherheit erwünscht wird.

Busverschlüsselungsprozessoren stellen aus diesen Gründen ein ausgesprochen interessantes Softwareschutzkonzept für manipulationssichere Systeme dar. Angesichts der Kreativität, die ein Angreifer bei seiner Analyse eines Sicherheitssystemes entwickeln kann, lässt sich die Sicherheit eines derartigen Prozessors praktisch kaum mit formalen Mitteln beweisen. Die effektive Sicherheit eines Schutzmechanismus kann letztendlich nur durch eine Reihe von ernsthaft motivierten Angriffsversuchen (*hostile reviews*) bestimmt werden. Auch fehlgeschlagene Angriffe sind in der Computersicherheitsforschung wertvolle Ergebnisse, da letztendlich nur sie Hinweise auf die wirkliche Sicherheit eines Systems geben. Daher werden wir uns im folgenden Kapitel ausführlich mit der Sicherheit eines kommerziell genutzten Busverschlüsselungsprozessors befassen.

2 Sicherheitsanalyse eines Busverschlüsselungsprozessors

2.1 Der Dallas Semiconductor DS5002FP Secure Microcontroller

Der von Dallas Semiconductor hergestellte Sicherheits-Microcontroller DS5002FP [Dal93] ist der derzeit meistgenutzte Mikroprozessor mit Busverschlüsselung. Er kommt in einer Reihe von Sicherheitsanwendungen zum Einsatz, bei denen mehr Speicherplatz benötigt wird, als in Einchip-Prozessoren zur Verfügung steht und in denen dennoch kein unbefugter Zugriff auf die gespeicherte Software möglich sein darf. Er wird insbesondere auch bei Anwendungen eingesetzt, in denen ein einfaches verschlüsseltes Auswechseln der Anwendungssoftware aus der Ferne über Datenleitungen wünschenswert ist. Beispiele für Einsatzgebiete sind Kreditkartenterminals für die Online- und Offline-Autorisierung von Bezahlungsvorgängen, Sicherheitsmodule in Pay-TV-Decodern sowie das Sicherheitsmodul in den Stromgutschein-Verkaufssystemen aus Abschnitt 1.1.3. Nach dem älteren DS5000 Prozessor [Col87] repräsentiert er bereits die zweite Generation von Busverschlüsselungsprozessoren dieses Herstellers. Er galt bislang als der sicherste kommerziell verfügbare Microcontroller für manipulationssichere Module.

Der DS5002FP basiert auf einem CPU-Kern, der kompatibel zu dem sehr weit verbreiteten 8-bit Microcontroller 8051 aus der MCS-51 Prozessorfamilie der Firma Intel ist [Int93]. Er verfügt, wie der 8051, über zwei getrennte 16-bit Adressräume für Programm- und Datenspeicher und kann damit bis zu 128 Kilobyte RAM ansteuern. Vier bidirektionale 8-bit Parallel-Schnittstellen stehen zum Anschluss von Peripherie zur Verfügung. Es existieren ferner zwei Taktzähler, zwei Interrupt-Eingänge, eine Watchdog-Schaltung, eine Reset- und Takterzeugungsschaltung sowie 128 Datenregister und eine Reihe von Steuerregistern. Der DS5002FP kann mit Frequenzen im Bereich 1–16 MHz betrieben werden und wird als SMT-Baustein in einem *plastic flat pack* Gehäuse mit 80 Pins geliefert.

Als externer Speichertyp sind ausschließlich batteriegepufferte statische RAM-Bausteine mit 8, 32 oder 128 Kilobyte vorgesehen. Die Aufteilung zwischen Programm- und Datenspeicherbereich kann softwaremäßig beliebig in vier-Kilobyte-Schritten verändert werden. Die Stromversorgung der SRAM-Bausteine erfolgt über den `VCCO`-Pin des Prozessors, der eine Schaltung für den automatischen Wechsel von der normalen Stromversorgung zum Batterie-Backup enthält. Der Prozessor verfügt über einen regulären 5 V Stromversorgungseingang `VCC` sowie einen Eingang `VBAT` für den Anschluss einer 3 V Lithiumbatterie. Sobald die `VCC`-Betriebsspannung abfällt, wird der Prozessortakt angehalten, der `VCCO`-Ausgang für die Versorgung der SRAM-Bausteine über eine Diode mit `VBAT` verbunden und der Prozessor in einen Energiesparzustand gebracht, in dem er typischerweise nur noch etwa 5 nA Strom verbraucht. Wenn die `VCC`-Versorgungsspannung wieder aktiviert wird wartet der Prozessor 21 504 Taktzyklen bis der Taktoszillator wieder stabil läuft und führt anschließend einen Reset durch.

Der DS5002FP wurde speziell für Sicherheitsanwendungen ausgelegt. Er besitzt ein 64-bit großes Schlüsselregister K, dessen Inhalt von einem Adressbusverschlüsseler EA, einem Datenverschlüssler ED und einem Datenbusentschlüssler ED^{-1} genutzt werden. Wie in Abbildung 1 dargestellt, hängt die Datenverschlüsselung ED nicht nur vom Schlüsselregister, sondern auch von der Adresse ab.

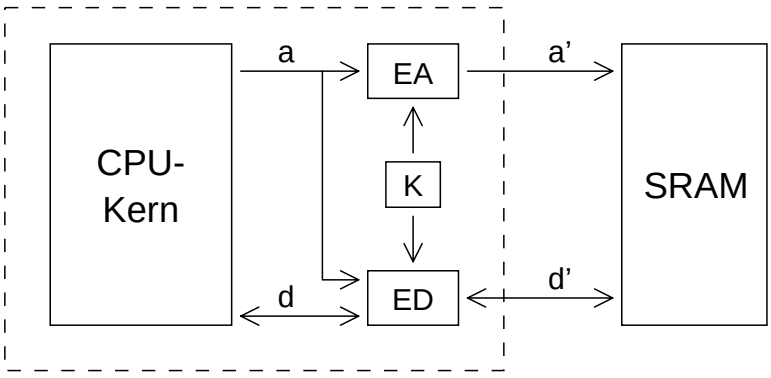


Abbildung 1: Die Busverschlüsselung im DS5002FP (gestrichelte Linie) bearbeitet einzeln jede Adresse und jedes Datenbyte, das zwischen externem RAM und CPU-Kern ausgetauscht wird.

Die Funktion ED ist ein Blockchiffre auf 8-bit Worten, während die Funktion EA ein Blockchiffre mit einer Wortgröße von 15-bit ist. Die obersten beiden Bits des 17-bit Adressbus sind unverschlüsselt, um verschiedene Speicherkonfigurationen aus 8 und 32 Kilobyte Speicherchips zuzulassen, was wir jedoch im Folgenden vernachlässigen können. Wenn der CPU-Kern einen Bytewert d an die Adresse a schreiben möchte, so wird im externen SRAM der Wert $d' = ED_{K,a}(d)$ in der Speicherzelle mit der Adresse $a' = EA_K(a)$ abgelegt. Bei einem Lesezugriff der CPU auf die Adresse a wird aus der externen Speicherzelle $a' = EA_K(a)$ der Wert d' geladen und nach der Entschlüsselung wird das Ergebnis $d = ED^{-1}_{K,a}(d')$ an den CPU-Kern zurückgegeben. Da sowohl EA als auch ED bijektive Funktionen sind, ist aus der Sicht des Programmierers kein Unterschied zu einem normalen unverschlüsselten RAM festzustellen. Anschaulich gesprochen sind alle Bytes im externen RAM untereinander durch die Adressverschlüsselung vertauscht worden und zusätzlich wird durch die Datenverschlüsselung jeder Bytewert einzeln mit einer eigenen adressabhängigen 256-elementigen Permutationstabelle verändert. Die Verschlüsselungsalgorithmen wurden vom Hersteller nicht dokumentiert und sind in [Dal93] lediglich als vereinfachte Varianten des DES-Algorithmus [FIP77] beschrieben.

Die ersten 48 Bytes des Programmspeichers befinden sich als sogenanntes Vektor-RAM auf dem Prozessorchip. Da das Vektor-RAM die Reset- und Interruptvektoren enthält wird so verhindert, dass der Angreifer die verschlüsselten Adressen der bekannten Einsprungpunkte nach einem Reset oder Interrupt beobachten kann. Während eines Zugriffs des CPU-Kerns auf das Vektor-RAM führt die Verschlüsselungslogik gleichzeitig einen Scheinzugriff auf eine durch einen Pseudozufallszahlengenerator erzeugte Adresse durch, damit der Angreifer, der den externen Bus beobachtet, keinen Anhaltspunkt dafür erhält, ob gerade auf den externen Speicher zugegriffen wird oder nicht.

Auch bei allen anderen Buszugriffszyklen, bei denen die CPU nicht auf den Hauptspeicher zugreifen muss, wird ein Scheinzugriff von der Kryptologik erzeugt, um einem Angreifer die Analyse der Vorgänge auf dem externen Bus zu erschweren. Der starre Befehlszyklus eines 8051-Prozessors dauert 12 Taktperioden und enthält immer zwei externe Buszugriffe. Ein Befehl kann ein oder zwei Befehlszyklen – also zwei oder vier Buszugriffe – zur Ausführung benötigen. Wenn aber ein Buszugriff vom gerade ausgeführten Befehl nicht benötigt wird, so holt der 8051-Prozessor schon einmal vorab das nachfolgende Instruktionsbyte, verwirft es aber beim anschließenden eigentlichen Instruktionsladezyklus wieder.

Der DS5002FP wandelt nun abhängig vom letzten Befehl alle bis auf einen dieser redundanten Zugriffe auf den nächsten Befehlscode in Scheinzugriffe um. Das heißt, es werden nicht nur unnötige Speicherzugriffe des 8051-Prozessors in pseudozufällige Scheinzugriffe verwandelt, sondern diese Scheinzugriffe werden eventuell auch mit dem nachfolgenden Instruktionsladezugriff vertauscht. Abbildung 2 verdeutlicht dies für einige verschiedene Instruktionsformate.

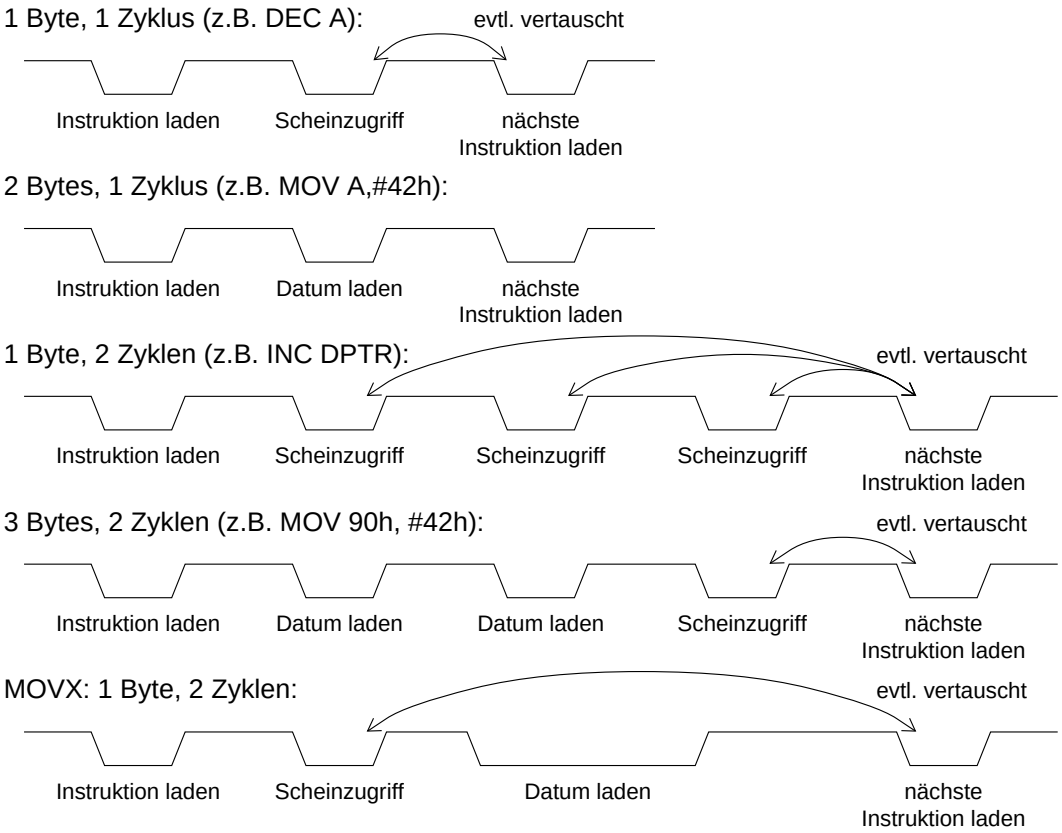


Abbildung 2: Dargestellt sind die Speicherzugriffsformen anhand der $\overline{\texttt{CE}}$-Leitung (*chip enable*) während der Ausführung verschiedener Befehle. Jeder Befehlszyklus entspricht zwei Speicherzugriffen (= $\overline{\texttt{CE}}$ low).

Der DS5002FP besitzt einen speziellen Selbstzerstörungseingang `SDI` (*self destruct input*). Sobald an diesem eine 5 V Versorgungsspannung anliegt, wird unabhängig von der normalen Versorgungsspannung das Schlüsselregister überschrieben, das interne Vektor-RAM gelöscht und die `VCCO`-Stromversorgung für die externen RAMs unterbrochen. An den `SDI`-Pin lässt sich der Alarmmechanismus einer zusätzlichen Sicherheitshülle um den Prozessor, den Speicher und weitere zu schützende Einrichtungen anschließen, der im Fall eines Eindringversuches sehr schnell und zuverlässig die geheime Anwendungssoftware vernichtet. Selbst wenn der Angreifer den verschlüsselten Inhalt der SRAM-Bausteine retten kann, nützt ihm dieser nichts ohne den bereits überschriebenen Schlüssel K im Prozessor.

Als weiteres besonderes Merkmal für Sicherheitsanwendungen verfügt der DS5002FP über einen Hardware-Zufallszahlengenerator, der beispielsweise für die Erzeugung von Schlüsseldaten und Zufallszahlen eingesetzt werden kann, wie sie in vielen kryptografischen Protokollen benötigt werden. Er wird durch zwei nicht sehr frequenzstabile Oszillatoren realisiert und kann alle 160 μs ein Zufallsbyte erzeugen, also 6250 Zufallsbytes pro

Sekunde.

Es ist auch eine Version DS5002FPM lieferbar, die über eine zusätzliche Metallisierungsschicht auf dem Siliziumchip verfügt. Diese Schicht besteht aus zwei gewundenen Leiterbahnen, die außer den Anschlusskontakten für die Bondingdrähte die gesamte Chipfläche abdecken. Diese Leiterbahnen sind mit einer Schaltung verbunden, von der sie ständig auf Unterbrechungen und Kurzschlüsse getestet werden. Wenn diese Schaltung anspricht, werden sofort die gleichen Operationen wie bei einer Spannung am Selbstzerstörungspin angestoßen. Damit soll verhindert werden, dass ein Angreifer mit Microprobing-Techniken und ähnlichen Chiptestverfahren, wie in Abschnitt 1.2.1 beschrieben, das Schlüsselregister oder die internen Systembusse abhören kann. Die Entwicklung der DS5002FPM-Version zeigt, dass der Hersteller Angriffe durch Chiptestverfahren als die schwächste Stelle des DS5002FP gesehen hat und mindestens einen MODH Schutzstandard erreichen wollte. Das Abdecken des batteriegepufferten Speichers durch ein metallisches Sensornetz dürfte ausgesprochen großen Aufwand bei einem Microprobing-Angriff verursachen. Dies gilt insbesondere, wenn schon die Ströme, die beim Auftragen von Zusatzleitungen mit fokusierten Ionenstrahlen durch den Angreifer auftreten, von der Alarmschaltung erkannt werden können.

Der Prozessor besitzt einen eingebauten Firmware-Monitor, der durch einen eigenen $\overline{\texttt{PROG}}$-Pin aktiviert werden kann. Über die serielle Schnittstelle kann der Anwender dieser eingebauten Software verschiedene Kommandos zum Konfigurieren des Speichers, zum Beschreiben, Auslesen und Verifizieren des Speicherinhaltes, usw. senden. Diese ROM-Software erzeugt beim Start automatisch mit Hilfe des Hardware-Zufallszahlengenerators einen neuen 64-bit Schlüssel, der vor dem ersten Speicherzugriffskommando nach dem Start in das Schlüsselregister K geladen wird. Mit dem Ladekommando `L` kann der Anwender eine Datei im Intel-Hex-Format an den Prozessor senden, der diese dann verschlüsselt im externen RAM ablegt. Nachdem mit dem Firmwarekommando `Z` ein spezielles Sicherheitsbit im Prozessor gesetzt wurde, stehen alle anderen Firmwarekommandos nicht mehr zur Verfügung, so dass über die Firmware kein Zugriff auf den unverschlüsselten Speicher möglich ist. Die einzige Ausnahme bildet das Kommando `U`, mit dem das Sicherheitsbit wieder gelöscht werden kann, wobei jedoch zuvor erst das Vektor-RAM und das Schlüsselregister gelöscht werden. Selbst wenn es dem Angreifer gelingen sollte, das Sicherheitsbit durch irgendeine Manipulation zu löschen, so könnte er dennoch nicht mit der Firmware die Software auslesen, da diese prinzipiell beim ersten Speicherzugriff nach einem Neustart zuerst einen neuen Zufallswert in K abspeichert.

2.2 Planung eines Angriffs

In [Abr91] werden Angriffe auf Sicherheitshardware in drei grobe Kategorien eingeteilt:

- **Mikroelektronischer Zugang.** Es wird versucht, direkt an die Hardwareelemente heranzukommen, die die geheimen Daten speichern oder verarbeiten.
- **Hardware-Simulation.** Wir ersetzen einen für uns zugänglichen Teil der Systemhardware durch eine Ersatzhardware und geeignete Software, mit der wir die Schutzmechanismen des Systems umgehen und in den geschützten Bereich eindringen können.

– **Abhören.** Wir nutzen die vom System an verschiedenen Stellen abgegebenen Strahlungen aus oder die genauen Zeitintervalle zwischen bestimmten beobachtbaren Ereignissen, um damit Rückschlüsse auf die im Innerenen verborgenen Vorgänge ziehen zu können.

Der mikroelektronische Zugang schied für meine Untersuchung aus, da in dem mir zur Verfügung stehenden Labor keine entsprechende Ausrüstung direkt zur Verfügung stand, und da der DS5002FPM gegen diese Angriffsart ausgesprochen gut gesichert erschien, wie schon frühere Sicherheitsanalysen dieses Prozessors ergaben [GEI94]. An der Universität Erlangen-Nürnberg sind aber prinzipiell geeignete Systeme und Labors vorhanden. Dazu würde beispielsweise ein Microprobing-Arbeitsplatz gehören, also ein spezielles optisches Mikroskop mit dem noch 0.5 μm feine Chipstrukturen erkennbar sind und dessen unterste Linse mindestens 5 cm Arbeitsabstand vom untersuchten Objekt hat sowie ein Satz feine Microprobing-Metallnadeln und entsprechende Befestigungen. Derartige Systeme kosten etwa 50 000 DM oder mehr. Ein gutes für mikroelektronische Zugänge ausgerüstetes Chiptest-Labor verfügt daneben über eine Plasmaätzanlage mit der selektiv einzelne Chipschichten in genau einer Raumrichtung entfernt werden können sowie eventuell auch ein Rasterelektronenmikroskop, einen Elektronenstrahltester und eine Focused-Ion-Beam-Workstation.

Der Abhör-Angriff könnte beispielsweise durch eine Analyse des genauen hochfrequenten Stromverbrauchs des Prozessors erfolgen. Es wäre theoretisch denkbar, eine Datenbank mit den exakten Stromverbrauchsmerkmalen jedes einzelnen Maschinenbefehls anzulegen, in der Hoffnung, damit den gerade ausgeführten Befehl identifizieren zu können. Durch die Fertigungstoleranzen der Chips variiert der Stromverbrauch jedoch bei den einzelnen Bausteinen. Ferner ist auch nicht klar, wie dies letztendlich zu einem erfolgreichen Angriff führen kann, da dann immer noch die im Prozessor verarbeiteten Daten unbekannt wären.

Ein alternativer Abhör-Angriff ergäbe sich aus einer Besonderheit des DS5002FP Prozessors: Er kann zum Anschluss von speicherabgebildeten Peripheriegeräten zwei der vier Parallelschnittstellen in externe unverschlüsselte (!) Systembusse verwandeln, über die alle Speicherzugriffe die über den Adressbereich des externen verschlüsselten RAMs hinausgehen abgewickelt werden. Dies bedeutet, dass im Prozessor eine kurze Verbindung zwischen dem internen unverschlüsselten Bus und einigen Schnittstellenpins besteht, die nur durch wenige Transistoren getrennt sind. Wenn wir Transistoren nicht als streng digitale Schalter sondern als analoge Elemente betrachten, so wäre es denkbar, dass die Werte auf den internen Bussen als winzige Spannungs- oder Stromschwankungen auf den Schnittstellenpins nachweisbar sind. Durch den Anschluss empfindlicher Messgeräte und optimal ausgewählter Lasten und Quellen an diese Pins könnten sich eventuell die Vorgänge auf den internen Bussen mithören lassen. Bei CMOS-Technologien entspricht das Schaltverhalten der Transistoren in der Tat fast dem von idealen digitalen Schaltern, aber bei in NMOS-Technik produzierten Schaltungen könnte dieser Ansatz durchaus erfolgreich sein. Aufgrund des zu erwartenden messtechnisch recht anspruchsvollen Versuchsaufbaus habe ich auch diesen Ansatz nicht weiter verfolgt.

Bleibt also die Simulation einer zugänglichen Systemkomponente. Im vorliegenden System kommt dafür nur eine Simulation der SRAM-Bausteine in Frage, da weitere Bestandteile – wie etwa die Firmware – entweder im geschützten Prozessorchip integriert sind, oder – wie die Takt- und Stromversorgung – für die Sicherheit wenig relevant erscheinen.

Die Idee dabei ist, dem Prozessor systematisch an jeweils derselben Adresse eine ganze

Reihe von verschiedenen Werten anzubieten und daraufhin seine Reaktion zu beobachten. Auf diese Art können die verschlüsselten Maschinencodes für einzelne Prozessorbefehle ermittelt werden. Diese einmal erkannten Befehle können dann genutzt werden um mehr Informationen zu gewinnen.

Für die Simulation des SRAMs benötigen wir eine Anordnung wie sie schematisch in Abbildung 3 dargestellt ist. In die normale Verbindung zwischen Prozessor und RAM wird eine Art Umschalter integriert. Durch diesen können die Daten, die der Prozessor bei einem Buslesezyklus anfordert, statt aus dem SRAM auch aus einem FIFO-Speicher bezogen worden. Ein Steuerrechner mit einer speziellen Steuerlogik, die zusammen den Angriff kontrollieren, bringen die untersuchte CPU in den Resetzustand und füllen den Befehls-FIFO mit einer mehrere Byte langen Testsequenz. Anschließend wird der Prozessor durch Freigeben des Reset-Signals gestartet. Wenn der Prozessor auf eine zuvor festgelegte SRAM-Adresse zugreifen will, um einen neuen verschlüsselten Befehlscode zu laden, dann wird auf den Befehls-FIFO umgeschalten. Der Prozessor liest nun statt der im SRAM abgespeicherten Anweisung eine Testsequenz aus dem FIFO und versucht diese zu entschlüsseln und auszuführen. Während dessen werden in weiteren FIFO-Speichern die Reaktionen der CPU auf dem Adressbus sowie auf einer der vier 8-bit Parallelschnittstellen festgehalten. Sobald der Befehls-FIFO entleert wurde hält die Steuerlogik die CPU an und informiert den Steuerrechner. Dieser liest die Adress- und Port-FIFOs, wertet die aufgezeichnete Reaktion der CPU aus und plant darauf basierend die nächste Testsequenz.

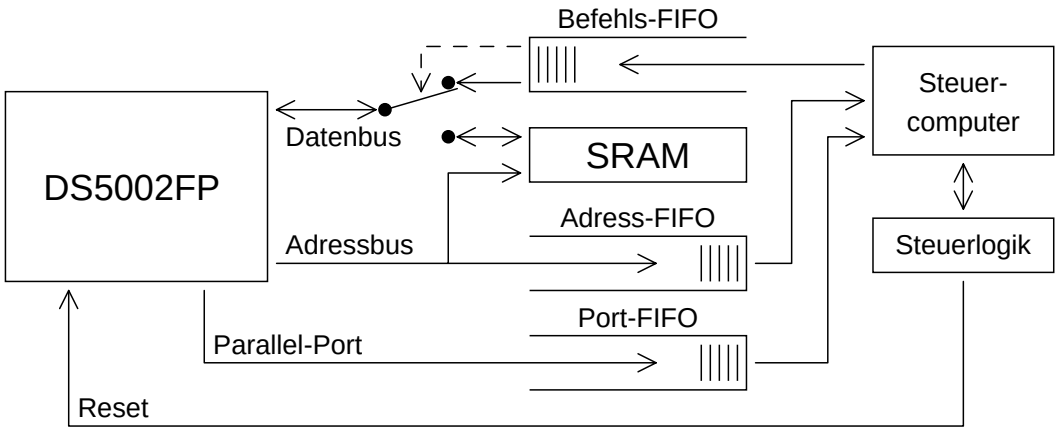


Abbildung 3: Mit der für den SRAM-Simulationsangriff benötigten Schaltung können die von der CPU geladenen SRAM-Bytes willkürlich ersetzt und die Reaktion der CPU darauf beobachtet werden.

Der Einbau des Busumschalters in ein bereits laufendes System lässt sich mit verhältnismäßig geringem Aufwand durchführen, sofern Zugang zur Systemplatine besteht. Wir müssen dazu nur die Verbindungen von den Prozessor-Ausgängen $\overline{\texttt{CE}}$ (*chip enable*) und R/$\overline{\texttt{W}}$ (*read/write*) zu den SRAM-Eingängen $\overline{\texttt{CS}}$ (*chip select*) und $\overline{\texttt{WE}}$ (*write enable*) durchtrennen. Auf alle Anschlüsse des Prozessorchips greifen wir mit einem geeigneten aufgesteckten Testclip zu, zum Beispiel mit dem PO5751 der Firma ITT Pomona (siehe Abbildung 4) der bei diesen Versuchen eingesetzt wurde oder dem CLIP-080-QF08-A der Firma Emulation Technology. Zusätzlich stellen wir durch Lötverbindungen einen Kontakt zwischen den $\overline{\texttt{CS}}$- und $\overline{\texttt{WE}}$-Pins jedes SRAM Bausteins und den entsprechenden Kontakten der Steuerlogik her. Nun kann die Steuerlogik wahlweise entweder die $\overline{\texttt{CE}}$- und R/$\overline{\texttt{W}}$-Signale des Prozessors unverändert an die SRAM-Chips weiterleiten, oder aber sie belässt die-

se beiden SRAM-Eingänge im *high*-Zustand und aktiviert den Befehls-FIFO. Wenn die $\overline{\texttt{CS}}$-Eingänge auf 5 V Pegel liegen, so verhalten sich die SRAM-Chips völlig passiv und alle Ausgänge sind hochohmig. Das $\overline{\texttt{CE}}$-Signal des Prozessors kann daher genutzt werden, um die Ausgangstreiber des Befehls-FIFOs zu aktivieren, der nun statt dem eigentlich angesprochenen SRAM-Baustein die Kontrolle über den Datenbus übernimmt. Aus dem $\overline{\texttt{CE}}$-Signal gewinnt zugleich die Steuerlogik die für die FIFOs benötigten Taktsignale.

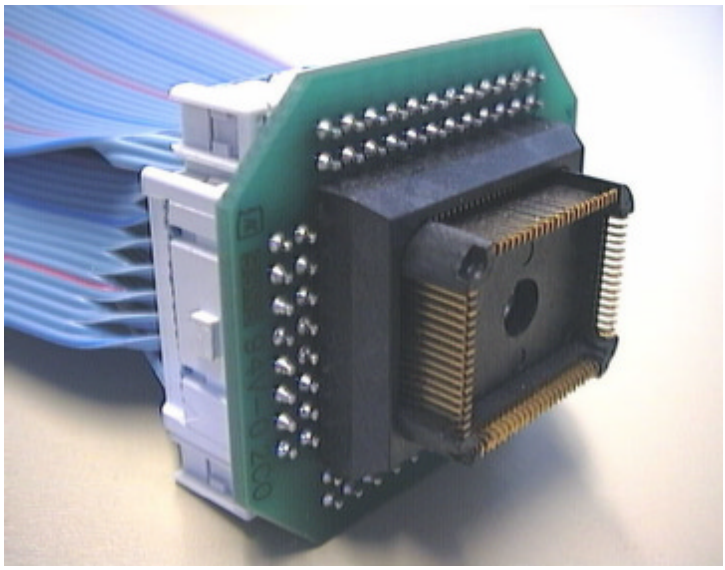


Abbildung 4: Testclip zum einfachen Kontaktieren aller Pins des DS5002FP.

Mit Hilfe der SRAM-Simulationsschaltung können wir dem untersuchten Prozessor in kurzer Zeit eine große Anzahl verschiedener Bytefolgen präsentieren und beobachten, wie er darauf reagiert. Wir suchen nach einigen ausgewählten Befehlsfolgen, die sich eindeutig anhand der Reaktion des Prozessors identifizieren lassen, und mit denen wir Informationen über die Datenbusverschlüsselungsfunktion ED gewinnen können.

Als geeignete Maschinenanweisung, nach deren verschlüsselter Representation wir suchen sollten, hat sich ein direkt adressierter Datentransferbefehl auf einen der Parallelports erwiesen. Zum Beispiel sorgt die 8051-Assembleranweisung

```
75 a0 42    MOV a0h, #42h
```

dafür, dass zwei Buszugriffe später der Wert hexadezimal `42h` auf den acht Pins der Parallelschnittstelle P2 anliegt. Natürlich könnten wir jederzeit auch nach einer `MOV`-Anweisung auf die Adresse einer der drei anderen Ports suchen, also in den folgenden Beispielen statt `a0h` auch `80h` (P0), `90h` (P1) oder `b0h` (P3) einsetzen. Wichtig ist nur, dass alle Pins der gewählten Parallelschnittstelle als Ausgänge arbeiten können. Alle 32 Schnittstellenpins des DS5002FP sind Ausgänge mit offenem Kollektor und die Schnittstellen P1–P3 haben schwache Pull-Up-Widerstände integriert, daher können alle Pins ohne Umschaltung als Ein- oder Ausgang genutzt werden. Ist jedoch ein Pin an den Ausgang einer Peripherieschaltung angeschlossen, so könnte er vom Prozessor nicht mehr als Ausgang benutzt werden. Daher sollte vor dem eigentlichen Angriff unbedingt ein Schaltplan des untersuchten Sicherheitssystems erstellt werden, in dem die an den Portpins des DS5002FP hängenden Schaltungsteile ersichtlich sind. Entweder existiert ein als Ausgang frei benutz-

barer Parallelport, oder es müssen entsprechende Leitungen als Vorbereitung durchtrennt werden.

2.3 Tabellarisierung der Datenbusverschlüsselung

Um die verschlüsselte Darstellung der direkt adressierten `MOV`-Anweisung auf den ausgewählten Port zu finden, füllen wir den Befehls-FIFO mit den fünf Werten

$$X, Y, Z, \texttt{00h}, \texttt{00h}^*, \tag{T1}$$

wobei die Bytes X und Y die Laufvariablen einer Suchschleife darstellen, die alle 2^{16} Kombinationen für diese beiden Werte ausprobiert. Die gesuchte `MOV`-Instruktion benötigt zwei Befehlszyklen und führt daher vier Buszugriffe durch. Da die Anweisung nur drei Bytes lang ist und keine weiteren Daten benötigt ist einer dieser Buszugriffe ein Scheinzugriff. Versuche haben gezeigt, dass der beim vierten Buszugriff geholte Wert keinen Einfluss auf die Ausführung des Befehls hat und dass der auf den Port ausgegebene Wert dort erst erscheint, wenn bereits das nachfolgende Instruktionsbyte geladen wird. Das vierte Byte `00h` im FIFO dient also nur als Platzhalter für einen Scheinzugriff. Das fünfte Byte `00h` sorgt dafür, dass der Prozessor noch solange weiterläuft bis der eventuell ausgegebene Portwert abgelesen werden konnte, da nach Entleerung des FIFO der Prozessor sofort automatisch angehalten wird, um die unkontrollierte Ausführung von willkürlichen Befehlen auf das notwendige Minimum zu reduzieren. Die *-Markierung zeigt nur an, dass wenn der Prozessor dieses Byte aus dem Befehls-FIFO holt, der am ausgewählten Parallelport anliegende Wert P aufgezeichnet wird.

Wir testen für alle 2^{16} Wertepaare (X, Y), ob die dabei jeweils entstehende Abbildung $E_0 : P \mapsto Z$ eine bijektive Funktion ist, also ob 256 verschiedene Werte für Z zwei Buszugriffe später 256 verschiedene Werte P auf dem ausgewählten Parallelport erscheinen lassen. Für den häufigen Fall, dass für eine (X, Y)-Kombination überhaupt keine Reaktion an den Schnittstellenpins sichtbar wird, reichen bereits zwei verschiedene Testwerte für Z aus, um die Bijektivität von E_0 auszuschließen. Daher müssen für diese erste Testrunde nicht viel mehr als 2^{17} CPU-Resets durchgeführt werden, weswegen dieser Suchlauf mit über 300 Resets pro Sekunde in wenigen Minuten abgeschlossen werden kann.

Während all diesen 2^{17} Einzelversuchen – und denen die noch folgen werden – kommt es darauf an, dass die Umschaltung vom SRAM zum FIFO jedesmal genau stattfindet, bevor die CPU das ersten Instruktionsbyte für jeweils dieselbe Anweisung aus dem SRAM holen will. So wird sichergestellt, dass die untersuchte CPU das erste FIFO-Byte X jedesmal scheinbar von derselben unverschlüsselten Adresse a_0 holt und daher die Datenbusentschlüsselungsfunktion ED^{-1}_{K,a_0} auf X anwendet.

Sobald wir das Paar (X, Y) mit $X = ED_{K,a_0}(\texttt{75h})$ und $Y = ED_{K,a_0+1}(\texttt{a0h})$ identifiziert haben, kennen wir auch alle 256 Einträge der Wertetabelle der Datenbusverschlüsselungsfunktion für die Adresse $a_0 + 2$, denn die gesuchte `MOV`-Anweisung stellt ja gerade sicher, dass der von $a_0 + 2$ geladene Wert entschlüsselt auf den Port ausgegeben wird, also dass $P = ED^{-1}_{K,a_0+2}(Z)$.

Leider ergibt sich jedoch das kleine Problem, dass der beschriebene Bijektivitätstest für E_0 nicht ausreicht, um $X = ED_{K,a_0}(\texttt{75h})$ eindeutig zu identifizieren. Auch die Maschinenanweisungen `XRL` ($X = ED_{K,a_0}(\texttt{63h})$), `ORL` ($X = ED_{K,a_0}(\texttt{43h})$) und `ANL` ($X =$

ED_{K,a_0}(53h)) können die Abbildung E_0 bijektiv werden lassen. Diese drei Anweisungen verknüpfen den vorherigen Wert auf der Parallelschnittstelle mit den bitweisen boolschen Operationen *xor*, *or* oder *and*. XRL wird bei konstantem vorherigen Portwert immer eine Bijektion erzeugen die sich von der gesuchten unterscheidet, solange der vorherige Portwert nicht 00h war. ORL erzeugt nur 256 verschiedene Ausgabewerte wenn die von ORL nicht auf 1 gesetzten Bits bereits zuvor den Wert 0 hatten, und in diesem Fall entsteht die gleiche Abbildung wie bei der gesuchten Abbildung für MOV. Ähnlich verhält es sich mit ANL, nur dass hier für eine bijektive Abbildung die nicht auf 0 gesetzten Bits bereits zuvor den Wert 1 haben müssen. Da nur XRL eine von der gesuchten abweichende Abbildung erzeugen kann, genügt es, unter den insgesamt ermittelten Abbildungen eine Mehrheitsentscheidung zu treffen, um die von MOV erzeugte Abbildung E_0 zu identifizieren. Sollte diese Mehrheit nicht zustande kommen, zum Beispiel weil der alte Portwert konstant und ungleich ffh sowie 00h ist, so erzeugen nur MOV und XRL zwei verschiedene Abbildungen. Diese müssen dann beide gespeichert und gegebenenfalls in der nachfolgenden Testreihe ausprobiert werden.

Wir müssen um diese Mehrheitsentscheidung durchführen zu können alle 2^8 Werte für X durchlaufen. Da jedoch alle vier Maschinenbefehle MOV, XRL, ORL und ANL, die eine Bijektion in E_0 verursachen könnten, die Portadresse im zweiten Befehlsbyte stehen haben, können wir den Wert Y festhalten, sobald das erste Paar (X, Y) eine Bijektion erzeugte. Damit beschleunigt sich die Suche nach den verbleibenden X-Werten um den Faktor 256.

Nachdem wir, wie eben beschrieben, die Funktion $ED_{K,a_0+2} = E_0$ vollständig als Tabelle in Form der von MOV erzeugten Abbildung dargestellt haben, werden wir nun ähnlich auch die Datenbusverschlüsselungsfunktion für die Adressen a_0+3 bis a_0+9 ermitteln. Daneben suchen wir noch nach zwei verschlüsselten Befehlscodes N_0 und N_1, die von den Adressen a_0 und $a_0 + 1$ geladen werden und jeweils irgendeinen Einbyte-Befehl repräsentieren, der nur einen Befehlslade- und einen Scheinzugriff durchführt und keinen Einfluß auf die weiteren Befehle nimmt. Beispiele für geeignete Befehle sind etwa NOP (*no operation*), INC A oder SETB C.

Für die nächste Testreihe füllen wir den Befehls-FIFO jedesmal mit den sieben Bytes

$$X, Z/8, Y, E_0(\texttt{a0h}), Z, \texttt{00h}, \texttt{00h}^*. \tag{T2}$$

Wie zuvor sind auch hier X und Y die Laufvariablen der Suchschleife. Ziel dieser Testreihe ist es, eine NOP-Anweisung oder einen anderen vergleichbaren Einbyte-Befehl, gefolgt von der bereits zuvor gesuchten MOV-Anweisung zu finden. Durch die vorangestellte NOP-ähnliche Anweisung wird der MOV-Befehl von der nächsthöheren Adresse eingelesen. Dadurch wird Z von $a_0 + 3$ geladen, der Adresse für die wir in dieser Testrunde die Datenbusverschlüsselung tabellarisieren wollen.

Die verschlüsselte Zieladresse der MOV-Anweisung wird von der Adresse $a_0 + 2$ geladen. Da wir dafür bereits die Verschlüsselungsfunktion tabellarisiert haben, können wir den Wert E_0(a0h) ohne Versuche sofort korrekt einsetzen. Deshalb liegt die Komplexität der Suche in diesem Testlauf immer noch bei 2^{17}, obwohl wir diesmal nach einer Kombination aus zwei Maschinenbefehlen suchen. Für den durch X repräsentierten Einbyte-Befehl existieren sehr viele mögliche Kandidaten, und der erste gefundene reicht völlig aus und wird als N_0 abgespeichert. Daher benötigt die zweite Testreihe in der Regel weniger als 2500 Resets, die in wenigen Sekunden durchgeführt werden können. Wir suchen nach einem Paar (X, Y), welches zwei Bedingungen erfüllt. Zum einen muss – entsprechend wie

bereits bei der ersten Testreihe – die Abbildung $E_1 : P \mapsto Z$ bijektiv sein. Zusätzlich muss die verschlüsselte Adresse, von der die CPU das vierte FIFO-Byte $E_0(\texttt{a0h})$ holt, identisch mit der verschlüsselten Adresse $EA_K(a_0 + 2)$ sein, von der im ersten Testlauf T1 bei der Ermittlung der Tabelle E_0 der Wert Z geholt wurde. Durch diese Zusatzbedingung wird sichergestellt, dass X wirklich eine genau ein Byte lange Anweisung ist und dass daher die neue Tabelle E_1 wirklich der Funktion ED_{K,a_0+3} entspricht und dass der zweite FIFO-Wert $Z/8$ lediglich einen Scheinzugriff befriedigt. Wir geben der CPU während des Scheinzugriffs einen veränderlichen Wert wie beispielsweise $Z/8$, um sicherzustellen, dass X nicht eine der Anweisungen ist, die den Scheinzugriff mit dem Laden des nächsten Instruktionsbytes vertauscht. Der Wertebereich von Z wurde hier durch eine Division eingeschränkt, um nicht eine zweite Quelle für eine bijektive Ausgangsabbildung zu liefern, die vielleicht durch eine völlig andere Befehlskombination entstehen könnte.

Sollte die Testreihe T2 fehlschlagen und kein (X, Y)-Paar gefunden werden das den obigen Testkriterien genügt, und sollten Aufgrund der `MOV`/`XRL`-Zweideutigkeit von der ersten Testreihe T1 noch zwei Alternativen für die Tabelle E_0 zur Verfügung stehen, so wird nun die andere Alternative zur Bestimmung von $E_0(\texttt{a0h})$ herangezogen und die Testreihe T2 wiederholt. Auch für T2 taucht wieder das Problem auf, dass mehrere alternative Y-Kandidaten auftauchen können. Hier wird zur Lösung ebenso wie bei der ersten Testreihe unter allen E_1-Kandidaten eine Mehrheitsentscheidung durchgeführt um die von `MOV` erzeugte Tabelle zu identifizieren. Sollte dies nicht möglich sein, so werden gegebenenfalls ebenso beide E_1-Kandidaten im nächsten Testlauf ausprobiert.

Um die Tabelle $E_2 = ED_{K,a_0+4}$ zu erstellen und N_1 zu ermitteln, stellen wir vor die `MOV`-Anweisung noch ein Einbyte-Kommando, dass heißt wir füllen für die nächste Testreihe den Befehls-FIFO mit den neun Werten

$$N_0, \texttt{00h}, X, Z/8, E_0(\texttt{75h}), E_1(\texttt{a0h}), Z, \texttt{00h}, \texttt{00h}^* \qquad \text{(T3)}$$

und der erste erfolgreiche X Wert wird als N_1 festgehalten. Diese dritte Testreihe benötigt weniger als eine Sekunde, da dank der bereits bekannten Tabellen E_0 und E_1 nur noch ein Byte X mit 2^8 möglichen Werten gesucht werden muss und sich in der Regel schon unter den ersten zehn Versuchen ein Treffer befindet. Die restlichen 256 Resets sind notwendig, um die Tabelle $E_2 : P \mapsto Z$ zu bestimmen. Eine `MOV`/`XRL`-Mehrdeutigkeit kann hier nicht mehr entstehen, da wir den verschlüsselten Befehlscode $E_0(\texttt{75h})$ für `MOV` nicht erraten sondern direkt bestimmt haben.

Ab der nun folgenden Testreihe für die Tabelle $E_3 = ED_{K,a_0+5}$ werden keine Befehlscodes mehr gesucht, sondern nur noch mit bereits vorhandenen Tabellen der Datenbusverschlüsselungsfunktion verschlüsselt. Der Befehls-FIFO wird dazu mit den Werten

$$N_0, \texttt{00h}, N_1, \texttt{00h}, E_0(\texttt{00h}), E_1(\texttt{75h}), E_1(\texttt{75h}), E_2(\texttt{a0h}), Z, \texttt{00h}, \texttt{00h}^* \qquad \text{(T4)}$$

gefüllt, wobei $E_0(\texttt{00h})$ der verschlüsselte Maschinencode einer `NOP`-Anweisung ist. Da wir uns nicht darauf verlassen wollen, dass `NOP` nicht seinen Scheinzyklus mit dem nachfolgenden Befehlsladezyklus vertauscht, bieten wir dem Prozessor einfach zu beiden Gelegenheiten den nachfolgenden Befehlscode $E_1(\texttt{75h})$ an. Diese Testreihe benötigt nur noch genau 256 Resets, um alle Tabelleneinträge von E_3 zu bestimmen. Die weiteren Tabellen E_4 bis E_7 für die Adressen a_6 bis a_9 werden mit entsprechenden FIFO-Testsequenzen bestimmt, bei denen mit einer `NOP`-Anweisung das `MOV`-Kommando jeweils um eine Adresse weitergeschoben wird.

Der gesamte Programmadressbereich wird durch die Bussteuerung des DS5002FP nach Verlassen der Firmwareladeroutine automatisch schreibgeschützt. Daher können die zahlreichen unbekannten Maschinenbefehle, die während der Testläufe zur Erstellung der Tabellen E_0, E_1 und E_2 unkontrolliert ausgeführt werden, keine Teile der geheimen Software überschreiben. Dies beinhaltet ebenfalls das Vektor-RAM auf dem CPU-Chip. Der Datenadressraum könnte zwar durch ein entsprechendes Kommando während eines Testlaufs beschrieben werden, jedoch sorgt die Steuerlogik nach der Umschaltung auf den FIFO dafür, dass keine Schreibanweisungen mehr an das SRAM weitergeleitet werden.

Der 128 Byte große Registerbereich auf dem CPU-Chip dagegen wird während der Suchläufe mit hoher Wahrscheinlichkeit überschrieben. Jedoch lagert dort ein sorgfältiger Programmierer in der Regel keine Daten, die einen Reset überdauern müssen. Diese Daten sind auch bei einem normalen Prozessorabsturz vor dem Überschreiben gefährdet, da jederzeit beispielsweise eine Unregelmäßigkeit in der Versorgungsspannung, eine Spannungsspitze auf einem Eingangspin, oder gar ionisierende Strahlung die Ausführung von unvorhergesehenen Befehlen auslösen kann. Auch der 128 Byte große Adressbereich der Steuerregister ist durch ein Überschreiben gefährdet, jedoch werden die meisten Steuerregister bei einem Reset auf Voreinstellungswerte zurückgesetzt. Die Steuerregister, die längerfristig zuverlässig Daten halten müssen und beispielsweise die Speicherkonfiguration festlegen sind durch die Bussteuerung der CPU besonders geschützt. Sie können erst beschrieben werden, wenn zuvor in ein spezielles anderes *timed-access*-Register die beiden Werte `aah` und `55h` geschrieben wurden, und auch dann ist ein Schreibzugriff auf diese besonders kritischen Steuerregister nur für wenige Taktzyklen möglich. Schon alleine die Kürze der in den Suchläufen benutzen Testsequenzen verhindert, dass der *timed-access*-Mechanismus zum Beschreiben der Steuerregister aus Versehen ausgelöst wird. Gelegentlich findet eine Suchschleife einen Schreibzugriff auf das Steuerregister, das den Prozessor in einen Stromsparmodus versetzt, in dem der Prozessor ohne weitere Buszugriffe auf einen Interrupt wartet. Die Angriffssteuersoftware bricht daher einen Einzeltest ab wenn nach wenigen Sekunden der Befehls-FIFO immer noch nicht geleert wurde, da offensichtlich die Testsequenz den Stromsparmodus aktiviert hat.

2.4 Die Wahl des SRAM/FIFO-Umschaltzeitpunktes

Ein noch zu klärendes Problem bei der geschilderten Vorgehensweise bleibt die Bestimmung des Umschaltzeitpunktes. Wir können nicht sofort nach einem Reset erwarten, dass der DS5002FP die aus dem FIFO gelieferten Bytes überhaupt beachtet, da er zunächst Befehle aus dem Vektor-RAM ausführt und dort unter Umständen einige Zeit verweilen kann, bevor er die erste Instruktion aus dem externen SRAM läd. Bis dahin sieht ein Beobachter des Busses nur Scheinzugriffe, die er aber nicht als solche erkennen kann.

Die einfachste Möglichkeit, den Umschaltzeitpunkt zu bestimmen besteht darin, t Buszugriffe nach dem Reset abzuwarten. Wir müssen verschiedene Werte für t ausprobieren, bis wir einen gefunden haben, der den Umschaltzeitpunkt direkt vor einen SRAM-Befehlslesezugriff legt, der nicht mit einem Scheinzugriff vertauscht wurde. Die Testsequenz T1 ist mit einem Suchaufwand von 2^{17} zu langsam, um schnell aussagen zu können, ob t geeignet gewählt wurde. Für diesen Test hat sich ein wesentlich schnellerer Suchlauf nach Sprunganweisungen sehr bewährt. Wir füllen den Befehls-FIFO mit den Werten

$$X, Z, \texttt{00h}, \texttt{00h}, \texttt{00h}^{+} \qquad \text{(T5)}$$

wobei X die Laufvariable einer Suchschleife ist und die $^+$-Markierung bedeutet, dass wir während dieses Lesezugriffs die ausgelesene Adresse A aufzeichnen. Wir können davon ausgehen, dass das Byte X vom Prozessor tatsächlich als Befehl decodiert und ausgeführt wurde, wenn wir ein X finden, so dass $Z \mapsto A$ eine injektive Funktion ist. In diesem Fall repräsentiert X eine Sprunganweisung, die Z als Teil der Zieladresse auswertet. Da meist nur zwei Z-Werte getestet werden müssen, um zu erkennen, dass keine Injektion vorliegt, müssen nur wenige hundert Resets für einen Testlauf durchgeführt werden.

Eine andere Alternative wäre, auf ein extern sichtbares Verhalten des Prozessors zu warten. Jeder Prozessor muss sich irgendwann einmal mit seiner Peripherie in Verbindung setzen, sonst würde er keinen Zweck erfüllen. Ein ausgewähltes Ereignis im Kommunikationsablauf mit der Peripherie könnte als Auslöser für die SRAM/FIFO-Umschaltung herangezogen werden, da der Prozessor ganze Kommunikationsprotokolle sicher nicht im Vektor-RAM abhandeln kann und daher sehr wahrscheinlich zu diesem Zeitpunkt aus dem externen SRAM seine Befehle läd.

Auch der Adressbus könnte als Triggerquelle für die Umschaltung herangezogen werden. Ein einfacher Adressbus-Komparator, der jeweils beim ersten Auftauchen einer willkürlich gewählten Adresse $EA_K(a_0)$ während eines Buslesezyklus umschaltet, ist aber ungeeignet. Die Adresse $EA_K(a_0)$ könnte auch schon vor dem ersten Zugriff auf a_0 während eines Scheinzugriffs ausgegeben werden. Eine Adressbustriggerschaltung müsste daher auf eine charakteristische Sequenz von mehreren Adressen warten. Eine derartige Sequenz kann unter den Adressen ausgewählt werden, die während eines normalen Ablaufs des untersuchten Systems bereits beobachtet wurden. Eine Abfolge von drei oder vier Adressen als Triggerbedingung macht das zufällige Auslösen durch Scheinzugriffe sehr unwahrscheinlich.

2.5 Auslesen des geschützten Speichers

Mit den Werten N_0 und N_1 sowie den Tabellen E_0 bis E_7 haben wir alle notwendigen Informationen, um den Befehls-FIFO mit verschiedenen winzigen Programmen zu füllen, die sukzessive jede der geschützten Software zugängliche Information über den Parallelport unverschlüsselt nach außen liefert.

Um an den Inhalt des Programmadressraums zu gelangen, benutzen wir die Befehlsfolge

```
00          NOP
00          NOP
90 xx yy    MOV DPTR, #xxyyh
e4          CLR A
93          MOVC A, @A+DPTR
f5 a0       MOV a0h, A
00          NOP
```

die das an der Adresse `xxyyh` im Programmbereich gespeicherte Byte auf den Parallelport ausgibt. Wenn wir diese Maschinencodes verschlüsseln und Füllbytes für die Scheinzugriffe hinzufügen, so erhalten wir den folgenden Inhalt des Befehls-FIFOs:

N_0	`NOP`
`00h`	Scheinzugriff

N_1	`NOP`
`00h`	Scheinzugriff
E_0(`90h`)	`MOV DPTR, ...`
$E_1(X)$	Oberes Adressbyte
$E_2(Y)$	Unteres Adressbyte
E_3(`e4h`)	Scheinzugriff
E_3(`e4h`)	`CLR A`
E_4(`93h`)	Scheinzugriff
E_4(`93h`)	`MOVC A, @A+DPTR`
E_5(`f5h`)	Scheinzugriff
E_5(`f5h`)	Scheinzugriff
—	Lesezugriff zum SRAM durchschalten
E_5(`f5h`)	`MOV ..., A`
E_6(`a0h`)	Portadresse
E_7(`00h`)*	`NOP`

(T6)

Wie bereits erläutert, repräsentieren das 1. und 3. Byte nicht unbedingt genau die `NOP`-Anweisung, sondern es können auch andere hier äquivalente Einbyte-Anweisungen sein, die nur die Aufgabe haben, den Befehlszähler in den Adressbereich ab $a_0 + 2$ zu bringen, für den wir die Verschlüsselungsfunktion tabellarisiert haben. Der DS5002FP kann zwar Scheinzugriffe mit nachfolgenden Befehlsladezugriffen vertauschen, aber dieser Sicherheitsmechanismus lässt sich dadurch umgehen, dass wir einfach bei jedem Scheinzugriff bereits den Befehlscode der nachfolgenden Anweisung anbieten.

Während des 14. Zugriffs auf den FIFO, also beim vierten Buszugriff der `MOVC`-Anweisung, liefert die Steuerlogik nicht einen Wert aus dem Befehls-FIFO an die CPU zurück. Ein spezieller Steuercode im Befehls-FIFO an dieser Stelle sorgt dafür, dass der Zugriff an das SRAM weitergeleitet wird. Mit diesem Mechanismus kann ein Programm, das aus dem Befehls-FIFO heraus ausgeführt wird, immer noch auf die Daten des SRAM-Speichers im untersuchten System zugreifen. Wir benutzen den DS5002FP selbst, um die Daten aus dem SRAM auszulesen. Da der Adressbus der CPU niemals hochohmig wird, können wir nicht einfach von außen her die SRAMs auslesen, ohne entweder den 17-bit breiten Adressbus aufzutrennen oder einen Kurzschluss der Bustreiber in der CPU in Kauf zu nehmen. Beide Alternativen würden die Wahrscheinlichkeit eines versehentlichen Zerstörens der Information im untersuchten System erhöhen. Daher ist der Zugriff auf die SRAMs über die CPU eine sehr elegante Lösung, zumal dabei gleich die CPU die Datenentschlüsselung für uns vornimmt.

Den Datenadressraum lesen wir mit einer sehr ähnlichen Befehlsfolge

```
00          NOP
00          NOP
90 xx yy    MOV DPTR, #xxyyh
e0          MOVX A, @DPTR
f5 a0       MOV a0h, A
00          NOP
```

aus, die entsprechend das an der Adresse `xxyyh` im Datenbereich gespeicherte Byte auf den Parallelport ausgibt. Die verschlüsselten Maschinencodes und Füllbytes für die Scheinzugriffe für den Befehls-FIFO lauten:

N_0	`NOP`

`00h`	Scheinzugriff
N_1	`NOP`
`00h`	Scheinzugriff
E_0(`90h`)	`MOV DPTR, ...`
$E_1(X)$	Oberes Adressbyte
$E_2(Y)$	Unteres Adressbyte
E_3(`e0h`)	Scheinzugriff
E_3(`e0h`)	`MOVX A, @DPTR`
E_4(`f5h`)	Scheinzugriff
—	Lesezugriff zum SRAM durchschalten
E_4(`f5h`)	`MOV ..., A`
E_5(`a0h`)	Portadresse
E_6(`00h`)*	`NOP`

(T7)

Um die Steuerregister auszulesen, die Informationen über die Speicherkonfiguration enthalten, genügt eine sehr einfache Befehlsfolge:

```
00          NOP
00          NOP
85 a0 xx    MOV a0h, xxh
00          NOP
```

Die entsprechende Bytefolge für den Befehls-FIFO ist

N_0	`NOP`
`00h`	Scheinzugriff
N_1	`NOP`
`00h`	Scheinzugriff
E_0(`85h`)	`MOV ..., A`
$E_1(X)$	Adressbyte
E_2(`a0h`)	Portadresse
E_3(`00h`)	Scheinzugriff
E_3(`00h`)*	`NOP`

(T8)

Die hier dargestellten Befehlssequenzen erlauben es normalerweise, mehrere hundert Bytes pro Sekunde auszulesen. Es könnte aber sein, dass der Angreifer es sehr eilig hat, oder dass die Umschaltung vom SRAM auf den FIFO erst einige Zeit nach einem Reset erfolgen kann, weshalb nicht über 300 Resets pro Sekunde möglich wären. In diesem Fall könnte man auch längere Auslesesequenzen benutzen, die pro Reset gleich hundert Bytes auf der Parallelschnittstelle ausgeben und die Auslesegeschwindigkeit entsprechend beschleunigen. Dazu müssten bei der Tabellarisierung der Datenbusverschlüsselung über E_7 hinaus noch weitere Tabellen erstellt werden, um anschließend einen längeren FIFO-Inhalt verschlüsseln zu können.

2.6 Kryptografische Qualität der Verschlüsselungsalgorithmen

Ein n-bit Blockchiffrealgorithmus $E_K(P) = C$ ist ein Verfahren, das bijektiv einen unverschlüsselten n-bit Block P (*plaintext*) auf einen verschlüsselten n-bit Block C (*ciphertext*) abhängig von einem k-bit Schlüssel K abbildet. An ein gutes Blockchiffreverfahren werden

eine Reihe von Sicherheitsanforderungen gestellt, wobei davon auszugehen ist, dass der Angreifer alle Details des Algorithmus mit Ausnahme des leicht auswechselbaren Schlüssels kennt:

A1 Wenn dem Angreifer eine Reihe von (P, C)-Paaren mit $E_K(P) = C$ bekannt sind (*known plaintext attack*), so darf das Ermitteln von K oder das Schließen auf weitere (P, C)-Paare nicht viel einfacher sein als das systematische Durchtesten aller 2^k möglichen Schlüsselwerte K (*brute force attack*).

A2 Selbst wenn der Angreifer mehrfach die Möglichkeit hat, zu einem von ihm frei wählbaren C das Ergebnis $P = E_K^{-1}(C)$ (*chosen ciphertext attack*) oder zu einem frei wählbaren P das Ergebnis $C = E_K(P)$ (*chosen plaintext attack*) zu ermitteln, so darf er immer noch keine Möglichkeit haben, auf K oder auf weitere (P, C)-Paare schneller irgendwelche Rückschlüsse zu ziehen als durch systematisches Testen aller Werte von K möglich ist.

Wenn ein Algorithmus diese beiden Anforderungen erfüllt, so muss nur k so hoch gewählt werden (beim DS5002FP ist laut [Dal93] $k = 64$), dass 2^k Iterationen mindestens mehrere Jahrzehnte Rechenzeit auf den schnellsten verfügbaren Computern erfordern, um zu verhindern dass ein Angreifer durch Beobachten oder Testen der Verschlüsselung diese verstehen kann.

Gute Blockchiffre-Algorithmen, die die Anforderungen A1 und A2 erfüllen, müssen mindestens eine Reihe von einfachen Eigenschaften aufweisen [Sch96]: Die Relation zwischen der Eingabe P und dem Ergebnis C darf nicht durchschaubar sein. Wenn die Eingabe irgend eine Folge von paarweise verschiedenen Blöcken ist, beispielsweise die Folge von Binärzahlen $0, 1, 2, 3, \ldots$, dann muss das Ergebnis der Verschlüsselung immer eine Folge von Blöcken sein, die nach allen erdenklichen statistischen Test wie Zufallsbitfolgen aussehen. Wird auch nur ein einzelnes Bit im P geändert, so sollten sich im Schnitt die Hälfte aller Bits in C ändern. Wenn Bits in P geändert werden, so darf es keinen erkennbaren Zusammenhang mit den Bits geben, die sich in C ändern. Sind diese einfachen Eigenschaften schon nicht erfüllt, so sind die Anforderungen A1 und A2 höchstwahrscheinlich auch nicht erfüllt.

Die in den vorangegangenen Abschnitten beschriebenen Techniken erlauben es, sämtliche Sicherheitsmechanismen des DS5002FP zu umgehen, da Dank der sehr kleinen Blockgröße $n = 8$ der Datenbusverschlüsselungsfunktion alle (P, C)-Paare ermittelt und gespeichert werden können. Eventuelle kryptografische Schwächen von ED und EA entsprechend den Anforderungen A1 und A2 spielten bei diesem Angriff gar keine Rolle. Es sind in den letzten 25 Jahren zahlreiche Verschlüsselungsalgorithmen und entsprechende mathematische Grundlagen entwickelt worden, mit denen diese Anforderungen erfüllbar sind. Kryptografisch gute Funktionen EA und ED in einem Busverschlüsselungsprozessor zu implementieren ist daher nur eine Frage der vorhandenen Chipfläche und Signallaufzeit sowie der kryptografischen Kompetenz der Entwickler.

Da eine kryptografische Analyse der Funktionen ED und EA für die Abschätzung der Gesamtsicherheit angesichts des bereits beschriebenen sehr wirkungsvollen Angriffs garnicht entscheidend ist, werfen wir nur einen kurzen Blick darauf.

Die gesamte Wertetabelle der Funktion EA kann während des Auslesens des Speichers mit aufgezeichnet werden. Dazu muss nur die durch die Befehlssequenz ausgelesene Adresse

zusammen mit der im Adress-FIFO aufgezeichneten Adresse während des an den SRAM weitergeleiteten Lesezugriffs in einer Tabelle notiert werden. Auch die Funktion ED kann mit fast dem gleichen Befehls-FIFO-Inhalt vollständig tabellarisiert werden. Dazu wird nicht mehr der Lesezugriff an das SRAM durchgestellt, sondern der entsprechende Lesezugriff wird nacheinander mit allen 256 möglichen Werten aus dem Befehls-FIFO beantwortet. Der über den FIFO an die CPU gelieferte Wert wird zusammen mit der Antwort auf dem Port in einer Tabelle aufgezeichnet. Für die Erstellung einer ED-Tabelle müssen für alle 2^{17} Adressen jeweils 256 Resets durchgeführt werden, was fast einen ganzen Tag dauert.

2.6.1 Die Datenbusverschlüsselung

Die Untersuchung der tabellarisierten Datenbusverschlüsselungsfunktion $ED_{K,a}$ an einer festen Adresse a zeigte den folgenden Zusammenhang. Sei $ED_{K,a}(P) = C$ und seien p_i und c_i die Bits der Binärdarstellung von P und C, d.h.

$$P = \sum_{i=0}^{7} p_i 2^i \quad \text{und} \quad C = \sum_{i=0}^{7} c_i 2^i.$$

Beim Betrachten der Wertetabelle von $ED_{K,a}$ fällt auf, dass wenn sich in P ein Bit ändert, sich in C daraufhin immer die selben Bits ändern, unabhängig vom Wert der anderen Bits in P. Weitere Tests haben gezeigt, dass $ED_{K,a}$ als affin-lineare Abbildung im Vektorraum $\mathrm{GF}(2)^8$ dargestellt werden kann. Es existiert für jede Adresse a eine vom Schlüssel abhängige 8×8-bit Matrix $M = (m)_{i,j} \in \mathrm{GF}(2)^{8\times8}$ sowie ein Vektor $b \in \mathrm{GF}(2)^8$, so dass gilt

$$c_i = b_i + \sum_{j=0}^{7} m_{i,j} p_j.$$

Weitere Versuche haben gezeigt, dass die adressabhängige Matrix M sich darstellen lässt als das Produkt $M = S_i \hat{M}$ aus einer adressunabhängigen Matrix $\hat{M}$ und einer adressabhängigen Permutationsmatrix S_i (beide aus $\mathrm{GF}(2)^{8\times8}$). Der DS5002FP benutzt nur 64 verschiedene Permutationsmatrizen S_i, die von einer sehr einfachen Funktion u abhängig von der verschlüsselten Adresse ausgewählt wird. Auch der adressabhängige Vektor b ist eine einfache Funktion v der verschlüsselten Adresse. Damit lässt sich die Ver- und Entschlüsselung des Datenbusses darstellen in der Form

$$\begin{aligned} ED_{K,a}(P) &= v_K(EA_K(a)) + S_{u_K(EA_K(a))}\, \hat{M}\, P \\ ED^{-1}_{K,a}(C) &= \hat{M}^{-1} S^{-1}_{u_K(EA_K(a))}\, (C + v_K(EA_K(a))) \end{aligned}$$

Die „Einfachheit“ von u und v besteht darin, dass diese Funktionen mit sehr wenigen Logikgattern implementierbar sind. Beispielsweise lässt sich für einen willkürlich gewählten Schlüssel K die Funktion u_K als in C implementierte Funktion wie folgt dargestellen:

```
int u(int ca)
{
  if (ca & 128) ca ^= 0x7f00;
  if (ca & 4) ca ^= 3;
```

```
  return ((ca >> 7) & 0x3e) | (ca & 1);
}
```

Die Permutationsmatrizen lassen sich mit sehr wenig Chipfläche implementieren, wenn alle 64 Permutationen der Menge sich als das Produkt der Matrizen zweier wesentlich kleinerer Mengen darstellen lassen. Die Matrizen $\hat{M}$ und $\hat{M}^{-1}$ lassen sich als jeweils 64-bit große Schlüsselregister abspeichern. M muss regulär sein, damit ED eine bijektive Funktion ist.

Auch ohne dass hier alle Details der Implementation von ED erläutert sind sollte klar sein, dass die im DS5002FP verwendete Datenbusverschlüsselung keineswegs den Anforderungen genügt. Sobald dem Angreifer für eine Adresse a genügend (P, C)-Paare bekannt sind, kann er entsprechend b und M mit einfacher linearer Algebra ermitteln und kennt damit vollständig $ED_{K,a}$ für eine Adresse a. Daneben kennt er bis auf 64 Permutationsmöglichkeiten auch M für alle anderen Adressen. Da die verschlüsselte Adresse dem Angreifer bekannt ist, ist auch zu erwarten, dass die sehr einfachen Funktionen u und v anhand von einigen bekannten Beispielwerten schnell erraten werden können.

2.6.2 Die Adressbusverschlüsselung

Der Algorithmus, der hinter EA steckt, lässt sich nicht mehr ganz so einfach rekonstruieren wie bei ED. Die Hinweise auf den DES-Algorithmus [FIP77] im Datenbuch [Dal93] lassen erahnen, dass EA eine Abfolge aus mehreren Bitpermutationen und Anwendungen von Substitutionstabellen ist, wobei Zwischenergebnisbits mit Schlüsselbits modulo zwei addiert werden. Der genaue Algorithmus wird jedoch vom Hersteller geheimgehalten.

Eine sehr effektive Analysemethode für DES-ähnliche Algorithmen ist die von Biham und Shamir vorgestellte „differentielle Kryptoanalyse“ [Bih93, Sch96]. Dabei wird untersucht, wie Veränderungen in den Eingangsbits sich auf die Veränderungen der Ausgangsbits auswirken. Wir wählen eine Differenz ΔP und eine große Menge von Eingangsdatenpaaren (P_1, P_2) mit $P_1 \oplus P_2 = \Delta P$, wobei P_1, P_2 und ΔP Binärworte mit der Blockgröße der Chiffrefunktion sind und $\oplus$ repräsentiert die bitweise *xor*-Funktion, also die Addition und Subtraktion im endlichen Körper $\mathrm{GF}(2)^n$. Wenn wir alle P_1, P_2 verschlüsseln erhalten wir eine große Anzahl von Worten $C_1 = EA_K(P_1)$ und $C_2 = EA_K(P_2)$. Von allen Paaren (C_1, C_2) bilden wir die Differenz $C_1 \oplus C_2 = \Delta C$. Von einem guten Kryptoalgorithmus würden wir erwarten, dass die Eingangsworte und Ausgangsworte in keinem irgendwie erkennbaren Verhältnis miteinander stehen. Daher sollten auch die Differenz ΔP der Eingangsworte und die Differenz der Ausgangsworte ΔC in keinem erkennbaren Zusammenhang stehen. Wenn wir eine große Anzahl von Paaren (P_1, P_2) mit einem festen ΔP testen, so sollten die dabei entstehenden ΔC-Werte in etwa uniform über die Menge aller möglichen n-bit Worte verteilt sein. Es hat sich aber bei vielen in der Literatur diskutierten Kryptoalgorithmen gezeigt, dass für bestimmte sorgfältig ausgesuchte ΔP-Werte die entsprechende ΔC-Verteilung nicht sehr uniform ist und dass die Verteilung Rückschlüsse auf einzelne Schlüsselbits zulässt. Mit dieser Technik konnten Biham und Shamir für mehrere bislang als sicher geltende Algorithmen mit einem PC in wenigen Minuten durch einen *chosen plaintext attack* den Schlüssel ermitteln, also Anforderung A2 ernsthaft verletzten.

Die aufwendigen Details der dabei eingesetzten Verfahren zu beschreiben würde den Rahmen hier bei weitem sprengen [Bih93]. Jedoch können wir die Verteilung von ΔC an

einigen Beispielen untersuchen, um zu überprüfen ob wenigstens annähernd ein uniforme Verteilung vorliegt, wie wir sie von einem sicheren Verschlüsselungsverfahren fordern.

Es hat sich gezeigt, dass bei der im DS5002FP eingesetzten Verschlüsselungsfunktion ΔC hochgradig ungleichmäßig verteilt ist. Wählen wir beispielsweise $\Delta P = 000000000000001$, ändern also am Eingang jeweils nur das am wenigsten signifikante Adressbit, so erhalten wir für die Werte von ΔC für einen festen Beispielschlüssel K die folgende Verteilung:

$$
\begin{aligned}
&p(\Delta C = 000000000010000) = 8/32 \qquad && p(\Delta C = 000000001001000) = 2/32 \\
&p(\Delta C = 000000001011000) = 2/32 \qquad && p(\Delta C = 000101000111001) = 2/32 \\
&p(\Delta C = 000101001100001) = 1/32 \qquad && p(\Delta C = 000101001110001) = 1/32 \\
&p(\Delta C = 010000000000001) = 2/32 \qquad && p(\Delta C = 010000000010001) = 2/32 \\
&p(\Delta C = 010101000101000) = 1/32 \qquad && p(\Delta C = 010101000111000) = 1/32 \\
&p(\Delta C = 010101001110000) = 2/32 \qquad && p(\Delta C = 101101000100110) = 1/32 \\
&p(\Delta C = 101101000110110) = 1/32 \qquad && p(\Delta C = 101101001101110) = 2/32 \\
&p(\Delta C = 111101000100111) = 2/32 \qquad && p(\Delta C = 111101001101111) = 1/32 \\
&p(\Delta C = 111101001111111) = 1/32
\end{aligned}
$$

Von den insgesamt 32 768 möglichen Differenzen ΔC tauchen bei einem willkürlich gewählten Schlüssel K nur 17 verschiedene auf. Für einen Angreifer ist gerade die Eingangsdifferenz $\Delta P = 000000000000001$ sehr interessant, da wenn der Prozessor bei der sequentiellen Befehlsausführung langer sprungfreier Befehlsfolgen auf aufeinanderfolgende Adressen zugreift, dann unterscheidet sich jedes zweite Paar von nacheinander auftretenden Adressen nur im ersten Bit, wenn wir die Daten- und Scheinzugriffe vernachlässigen. Es ist daher anzunehmen, dass der Angreifer die Verteilung von ΔC für $\Delta P = 000000000000001$ relativ einfach durch reine Beobachtung des Busses erraten kann. Wenn der Verschlüsselungsalgorithmus dem Angreifer bekannt ist, dann kann er vermutlich aus der Art der ΔC-Verteilung Rückschlüsse auf einige der Schlüsselbits gewinnen. Für andere Eingangsdifferenzen wie etwa $\Delta P = 000000010000000$ existieren sogar noch weniger mögliche ΔC-Werte:

$$
\begin{aligned}
&p(\Delta C = 000000101010101) = 1/4 \qquad && p(\Delta C = 000010100000001) = 1/4 \\
&p(\Delta C = 100000010101111) = 1/4 \qquad && p(\Delta C = 100010011111011) = 1/4
\end{aligned}
$$

Es scheint also, dass weder die Daten- noch die Adressbusverschlüsselung des DS5002FP einem *known-plaintext*-Angriff standhalten werden, wenn dem Angreifer die Details des Algorithmus bekannt sind. Die sequentiellen Adresszugriffe während der Befehlsausführung oder während der Berechnung von Prüfsummen über Daten, der Ein-/Ausgabe von Datenblöcken oder der linearen Suche in Tabellen liefern dem Angreifer viele Möglichkeiten, die Bitdifferenzen zwischen den unverschlüsselten Adressen bei beobachteten Speicherzugriffen zu erraten. Daher rät [Dal93] den Programmierern, lange sprungfreie Programmsequenzen und sequentielle Speicherzugriffe nach Möglichkeit zu meiden.

Der Bericht einer früheren kommerziellen Sicherheitsanalyse des DS5002FP [GEI94], bei der im Gegensatz zur vorliegenden Untersuchung vom Hersteller Einblick in die komplette Schaltung des Prozessors gewährt wurde, bestätigt meine Annahme, dass EA einem *known-plaintext*-Angriff nicht standhalten würde.

3 Mögliche Gegenmaßnahmen und Sicherheitsverbesserungen

3.1 Software-Gegenmaßnahmen

Da der DS5002FP in einer Reihe von sicherheitskritischen Anwendungen eingesetzt wird und nicht in allen Fällen ein zusätzlicher Alarmmechanismus als ein von der Busverschlüsselung unabhängiger Schutz vorhanden ist, wäre es wünschenswert, durch geschickte Softwaremodifikationen die Gefahr des im vorangegangenen Kapitel dargestellten RAM-Emulations-Angriffs abzumildern. Insbesondere der Vektor-RAM bietet hierzu einige Ansatzpunkte, die Aufgabe für den Angreifer etwas zu erschweren. Jedoch erscheint eine wirklich befriedigende Lösung durch reine Softwaregegenmaßnahmen nicht erreichbar. Im Folgenden sind einige Ideen für Schutzmaßnahmen dargestellt sowie die Techniken mit denen der Angreifer sie gegebenenfalls umgehen kann. Es zeigt sich dabei, dass die Sicherheit eines manipulationssicheren Systems, also die geringe Wahrscheinlichkeit eines erfolgreichen Angriffs, und dessen Zuverlässigkeit, also die geringe Wahrscheinlichkeit eines frühen Systemversagens, teilweise gegeneinander gerichtete Entwurfsziele sind. Der RAM-Emulations-Angriff profitierte zweifellos von den Fehlervermeidungsmechanismen des DS5002FP, und einige der denkbaren Gegenmaßnahmen zur Verbesserung der Sicherheit werden durch eine reduzierte Zuverlässigkeit erkauft.

3.1.1 Prüfsummenverfahren

Um sich gegen Manipulationen des RAM-Inhaltes und -Verhaltens zu schützen, könnte ein Programmierer auf die Idee kommen, im Vektor-RAM einen Prüfsummenalgorithmus zu implementieren. Diese Routine würde den Maschinencode der gesamten Anwendungssoftware mit einer im Vektor-RAM abgelegten Prüfsumme vergleichen. Sollte die errechnete Prüfsumme nicht mit der gespeicherten übereinstimmen, so geht der Prozessor in eine Endlosschleife und wird das Vektor-RAM vor dem nächsten Reset nicht mehr verlassen.

Für diesen Fall verfügt die im Anhang vorgestellte Steuerlogik über einen Zähler, mit dem ein RAM-Test abgewartet werden kann. Bis zum Umschaltzeitpunkt hat der Prozessor Zugriff auf den Speicher und kann Schreib- und Lesetests durchführen. Der Angreifer muss den Zähler hinreichend groß einstellen, so dass der RAM-Test zum Zeitpunkt der SRAM/FIFO-Umschaltung vorüber ist. Eine zu große Wartezeit vor dem Umschalten reduziert die maximale Anzahl der Resets pro Sekunde und verlängert entsprechend die Testläufe. Daher könnte der Angreifer versuchen, die geringste Wartezeit zu ermitteln, nach der mit dem Testlauf T5 sich eine Reaktion der CPU auf die FIFO-Bytes zeigt.

Diese Versuche könnten für eine Falle genutzt werden. Sobald die Prüfsumme einmal nicht mehr stimmt, legt die Schutzroutine aus dem Vektor-RAM heraus ein Passwort aus mehreren Bytes in einigen der 128 Registern ab. Dieses Passwort wird nach jedem Reset von der Schutzroutine im Vektor-RAM geprüft. Ist es vorhanden, so heißt dies, dass nach einem der vorangegangenen Resets ein Speichermanipulationsversuch entdeckt wurde und der Prozessor geht im Vektor-RAM in eine Endlosschleife. Er ist damit zwar nutzlos, da er nie wieder auf den externen Speicher zugreifen wird und muss neu mit Software geladen werden, jedoch kann mit diesem Prozessor auch nicht mehr durch einen RAM-Emulationsangriff die Verschlüsselungsfunktion tabellarisiert werden.

Um sich gegen zufällige Lesefehler während der Prüfsummenberechnung zu schützen, kann das Passwort zusammen mit einem Zählerwert bei einer falschen Prüfsumme abgespeichert werden. Ist das Passwort bereits vorhanden, so wird nur der Zählerwert erniedrigt wenn der Prüfsummentest fehlschlägt. Ist der Zählerwert 0, so wird in die Endlosschleife gesprungen.

3.1.2 Zufällige Verzögerungen

Solange der Angreifer nur nach einer bestimmten Zahl von Buszyklen den FIFO-Modus aktiviert könnte eine Sicherheitsroutine nach dem Reset vor Verlassen des Vektor-RAMs eine zufällige Zeit in einer Warteschleife verbringen. Die Anzahl der Warteschleifendurchläufe kann mit dem Hardwarezufallszahlengenerator bestimmt werden. Damit wäre die Voraussetzung nicht mehr erfüllt, dass das erste FIFO-Byte nach jedem Reset von der gleichen Adresse geladen wird und die systematische Suche nach bestimmten Befehlscodes wird fehlschlagen.

Ein einfacher Ausweg für den Angreifer besteht darin, nicht mit einem Zähler, sondern durch einen charakteristischen Ausgabevorgang des Prozessors die SRAM/FIFO-Umschaltung zu aktivieren. Jeder Prozessor hat nur einen Sinn, wenn er irgendwelche Daten und Kommandos an Peripheriegeräte sendet. Wenn eine bestimmte Reaktion, zum Beispiel die Ausgabe des ersten Bytes eines Datenpakettyps, stets von der selben Maschinenanweisung an derselben Adresse ausgeführt wird, so kann dieses Byte auf dem Port als Triggerbedingung genutzt werden.

Der Anwendungsprogrammierer könnte dann versuchen, mehrere Kopien der Ausgaberoutine im Speicher an verschiedenen Adressen zu halten und mit Hilfe des Hardwarezufallszahlengenerators nach jedem Reset aussuchen, welche Routine dieses Mal benutzt wird. Der Angreifer wiederum hat aber im Adress-FIFO aufgezeichnet, von welcher verschlüsselten Adresse die Anweisung geladen wurde, die nach der Umschaltung ausgeführt wurde. Mit dieser Adresse kann er die verschiedenen Kopien der Ausgaberoutine unterscheiden. Der Testlauf wird solange wiederholt, bis er für eine Adresse alle Werte zusammen hat. Bei den weiteren Tests werden die Ergebnisse die an anderen Adressen aufgezeichnet wurden verworfen und der Versuch wird wiederholt. Dies kann zwar die Laufzeit des Angriffs deutlich verlängern, macht ihn jedoch nicht unmöglich.

In einigen Anwendungen ist es vielleicht sogar möglich, die Ausgaberoutine, die eine Reaktion erzeugt auf die der Angreifer triggern kann, komplett in den Vektor-RAM zu verlegen und vor Verlassen des Vektor-RAMs wird wieder eine Zufallswarteschleife ausgeführt, so dass dem Angreifer keine Synchronisationsinformation durch die Ausgabe geliefert wird. In diesem Fall kann aber der Angreifer immer noch den Adressbus als Triggerquelle benutzen.

3.1.3 Die Registerfalle

Bei der Suche nach den Bytes $X = ED_{K,a_0}(\texttt{75h})$ und $Y = ED_{K,a_0+1}(\texttt{a0h})$ lässt es sich nicht vermeiden, dass zuvor bereits Anweisungen ausgeführt werden, die wahllos in den Bereich der 128 Datenregister auf dem Prozessorchip schreiben. Der Anwendungsentwickler könnte in einigen der 128 Datenregister ein Passwort ablegen, das von einer Sicherheitsroutine im Vektor-RAM nach jedem Reset überprüft wird. Sollte das Passwort überschrieben worden sein, so begibt sich die Sicherheitsroutine in eine Endlosschleife im

Vektor-RAM. Auch nach einem Reset wird das Passwort immer noch zerstört sein, und der Prozessor wird ohne Neuprogrammierung oder anderweitige Wiederherstellung des Passwortes nie mehr ein vom externen Bus geholtes Byte ausführen und ist daher für weitere RAM-Emulationsangriffe nutzlos.

Um das Passwort nach dem Programmiervorgang erstmals in die Register zu bekommen, müsste noch vor der Registerpasswortabfrage im Vektor-RAM eine weitere winzige Routine bereitstehen. Sie erlaubt über die Eingabe eines anderen Passworts auf der Parallelschnittstelle unmittelbar nach dem Reset die Registerpasswortabfrage zu überspringen und anschließend das Registerpasswort erstmals zu setzten.

Solange der Angreifer nicht den Mechanismus mit dem das Passwort erstmals gesetzt wurde erraten kann, um ihn erneut zu benutzen, hat er kaum eine Chance nicht in diese Registerfalle zu laufen. Jedoch ist das Verfahren mit einem großen Nachteil verbunden: Das Registerpasswort kann auch sehr leicht bei einem normalen Absturz überschrieben werden, wie er beispielsweise durch Spannungsschwankungen oder elektromagnetische Störungen gelegentlich verursacht werden kann. In der Zeit bis zur Aktivierung des Watchdog-Mechanismus kann der unkontrolliert Befehle ausführende Prozessor durchaus auch das Registerpasswort zerstören. Danach kann auch der Watchdog-Mechanismus nicht mehr die Funktionsfähigkeit des Systems herstellen. Für Anwendungen, bei denen die Zuverlässigkeit entscheidend ist, scheidet die Registerfalle als Gegenmaßnahme daher aus.

3.1.4 Die Resetfalle

Eine noch einfachere Alternative zur Registerfalle besteht darin, im Vektor-RAM die Anzahl der seit Inbetriebnahme des Systems durchgeführten Resets mitzuzählen. Beim Überschreiten einer Schranke springt der Prozessor noch im Vektor-RAM in eine Endlos-Schleife und führt folglich nie mehr eine extern geladene Instruktion aus. Der in Kapitel 2 vorgestellte Angriff führt größenordnungsmäßig etwa 10^5 Resets durch. Wenn die Anwendungsumgebung es zulässt, die Lebensdauer der Software auf unter 10^4 Resets zu beschränken, bietet sich dies als Übergangslösung an. Wie schon bei der Registerfalle wird auch hier die Zuverlässigkeit des Systems im Interesse der Sicherheit reduziert.

3.2 Hardware-Gegenmaßnahmen

Das Sicherheitsproblem, das der RAM-Emulationsangriff im DS5002FP aufgezeigt hat, lässt sich nur durch Änderungen an der CPU befriedigend lösen.

3.2.1 Verschlüsselung größerer Blöcke

Die grundlegende Schwachstelle des DS5002FP liegt darin, dass die Datenbusverschlüsselungsfunktion nur auf 8-bit Blöcken arbeitet. Das Alphabet des Chiffres ist damit klein genug, um die Funktion als Tabelle praktikabel darstellen zu können. Die naheliegende Lösung zur Verbesserung der Sicherheit besteht folglich in einer Erhöhung der Blockgröße von ED auf mindestens 64-bit, um eine systematische Tabellarisierung auszuschließen. Die 8-bit Blockgröße im DS5002FP war bei der Entwicklung sicherlich dadurch bedingt, dass die 8051-Architektur 8-bit orientiert ist. Jede beliebige 8-bit Adresse kann Ziel eines

Datenzugriffs oder Sprungbefehls sein und muss damit unabhängig von vorangegangenen und nachfolgenden Speicherzugriffen auslesbar und beschreibbar sein.

Bei Prozessoren mit einem Instruktions- und Datencache werden jeweils ganze Cachezeilen auf einmal geladen und wieder abgespeichert. Hier würde sich anbieten, den Cache zusammen mit der Busverschlüsselungslogik auf dem CPU-Chip unterzubringen und eine Cachezeile von jeweils mindestens acht Byte auf einmal zu verschlüsseln. Damit würde die Komplexität für die Suche nach einer bestimmten Maschineninstruktion auf 2^{64} steigen und eine Tabelle der Verschlüsselungsfunktion würde 2^{64} Einträge benötigen.

Ein Cache in der CPU bietet darüber hinaus den Vorteil, dass er das Zugriffsmuster auf den externen Speicher verschleiert. Beim DS5002FP können Schleifen leicht anhand von sich periodisch wiederholenden Zugriffen auf die gleiche Adressfolge erkannt werden. Der Ausgang von Fallunterscheidungen mit bedingten Sprunganweisungen lässt sich ebenfalls anhand der beobachtbaren Adresszugriffsfolge unterscheiden. Ein Cache läd bei einer hinreichend kurzen Schleife die entsprechenden Cachezeilen nur beim ersten Durchlauf und die Buslogik kann für die restlichen Durchläufe Scheinzugriffe auf den externen Speicher durchführen. Ähnlich sind bedingte Sprünge in kleinen Schleifen nicht mehr unterscheidbar, da nach dem erstmaligen Ansprung beider Alternativen deren Befehle nicht mehr neu geladen werden. Voraussetzung ist natürlich, dass die Cachezeilen der Fallunterscheidungs-Alternativen sich nicht gegenseitig verdrängen. Aus diesem Grund sollte eine mehrfach-assoziative Cachearchitektur benutzt werden. Die Adressbusverschlüsselung in einem Busverschlüsselungssystem mit Cache kann sich auf die oberen Adressleitungen beschränken, da die Differenzierung der einzelnen Worte innerhalb einer Cachezeile durch die unteren Adressbits ohnehin nicht von außen beobachtbar ist.

Die Verschlüsselung ganzer Cachezeilen ist sicherlich die wichtigste Maßnahme zu einer wesentlichen Verbesserung der Sicherheit eines Busverschlüsselungssystems. Doch selbst dann steht dem Angreifer immer noch in Form des Buszugriffmusters etwas Information über die Abläufe der geschützten Software zur Verfügung. In der Literatur wurden bereits Busverschlüsselungsarchitekturen diskutiert, bei denen beweisbar außer dem Ein-/Ausgabeverhalten der Software und der benötigten Bearbeitungszeit überhaupt keine weitere Information über die geschützte Software in Form von Buszugriffsmustern nach außen dringt [Gol86, Ost89]. Diese Verfahren sind aber sehr aufwendig zu implementieren und erfordern in regelmäßigen Abständen zusätzliche Schreib- und Lesezugriffe auf den gesamten Speicher, was sie derzeit für den praktischen Einsatz noch nicht attraktiv macht.

Für Anwendungen in denen Microcontroller wie der DS5002FP eingesetzt werden, kann aber die durch einen Cache verursachte schwierige Vorhersehbarkeit der Ausführungszeit einzelner Befehle hinderlich sein, insbesondere wenn der Prozessor für Echtzeitanwendungen wie die Handhabung eines zeitkritischen Kommunikationsprotokolls oder digitale Signalverarbeitung eingesetzt wird. Ein Cache und eine Verschlüsselungslogik mit angemessener Blockgröße benötigt eine nicht zu vernachlässigende Chipfläche und erhöht damit die Kosten des Systems.

Daher sind auch einfachere Gegenmaßnahmen als ein ganzer Cache von Interesse, insbesondere für die Entwicklung eines weitgehend kompatiblen Nachfolgeprozessors des DS5002FP, mit dem die in Kapitel 2 vorgestellte Sicherheitslücke durch Austausch des Prozessors in vielen existierenden Sicherheitssystemen beseitigt werden kann.

3.2.2 Verlängerung kritischer Befehlsfolgen

Besonders nützlich erwies sich beim Entwurf des RAM-Emulationsangriffs, dass nur zwei Instruktionsbytes gefunden werden mussten, mit denen die Verschlüsselungsfunktion tabellarisieren werden konnte. Der Angriff wäre wesentlich zeitaufweniger geworden, wenn eine systematische Suche nach mindestens vier Bytes erforderlich gewesen wäre. Daher bieten sich verschiedene Maßnahmen an, die den Angriff erheblich erschweren können:

- Der direkt adressierte `MOV`-Befehl auf die vier Parallelportadressen könnte im Microcode des Prozessors ganz unterbunden werden. Der Programmierer müsste auf registeradressierte Befehle ausweichen und der Angreifer müsste nach längeren Befehlsfolgen suchen.
- Der direkt adressierte `MOV`-Befehl auf die vier Parallelportadressen könnte mit einem um zwei Bytes verlängerten Befehlscode versehen werden.
- Beim direkt adressierten `MOV`-Befehl liegt der zu schreibende Datenwert an letzter Stelle im Befehlsformat. Dies vereinfacht die Suche beim Weiterschieben der `MOV`-Anweisung zur Erstellung der Tabelle für die nächste Adresse erheblich. Wenn das Befehlsformat so geändert würde, dass der ausgegebene Wert vom Befehlscode und der Portadresse umrahmt wird, dann wäre nach dem Weiterschieben ein erneutes Suchen nach der Portadresse notwendig.
- Einige Spezialregister für die Einstellung der Speicherkonfiguration werden durch einen *timed access* Mechanismus vor versehentlichem Überschreiben geschützt. Dazu muss die Anwendung zuerst die beiden Bytewerte `aah` und `55h` in ein anderes Register schreiben, bevor der Schreibzugriff auf das fragliche Spezialregister für einige Befehlszyklen erlaubt wird. Mit diesem Mechanismus könnten auch die Prozessorschnittstellen, über die unverschlüsselte Daten den Prozessor verlassen könnten, vor einem Zugriff durch systematische Suche nach den verschlüsselten Befehlscodes geschützt werden.

All diese Vorschläge sind auch mit kleineren Nachteilen verbunden: Die 8051-Kompatibilität wird aufgegeben und die über die Prozessorschnittstellen maximal erreichbaren Datenausgaberaten reduzieren sich etwas.

3.2.3 Unabhängige Daten- und Programmverschlüsselung

Unter Umständen ist es einem Angreifer auch möglich, die Verschlüsselungsfunktion für eine Reihe von aufeinanderfolgenden Adressen ganz ohne die Suche nach passenden verschlüsselten Befehlscodes zu ermitteln. Viele Anwendungen besitzen Funktionen, die ein Datenpaket über die Schnittstelle einlesen und im Datenadressraum abspeichern. Der Angreifer kann das Abspeichern der Werte beobachten. Er sendet systematisch Testpakete an die Anwendungssoftware im untersuchten Prozessor. In diesen Paketen taucht irgendwann an jeder Stelle jeder Bytewert einmal auf. Damit kann er für den Speicherbereich in dem die Pakete abgelegt werden die Verschlüsselungsfunktion tabellarisieren. Beim DS5002FP sind die obersten beiden Adressbits nicht verschlüsselt und sie beeinflussen auch nicht die Datenverschlüsselung. Dadurch existiert im Programmspeicher ein Adressbereich, der den gleichen Verschlüsselungsfunktionen unterliegt wie der bereits tabellarisierte Datenbereich. Für diesen Bereich kann der Angreifer eine Auslesebefehlsfolge anfertigen und auf den FIFO umschalten wenn die CPU zur Befehlsausführung auf diesen Bereich zugreift.

Als Gegenmaßnahme sollte vor allen Dingen die Datenverschlüsselung nicht mit der Programmverschlüsselung identisch sein. Programmierer des DS5002FP sollten vorerst darauf achten, dass Programmadressraumbereiche, die der gleichen Verschlüsselung unterliegen wie ein von außen ladbarer Datenadressbereich, nicht zur Befehlsausführung angesprungen werden. Wenn der Anwendungsprogrammierer die Startadresse des Pufferbereiches in dem das empfangene Paket gehalten wird jedesmal zufällig bestimmt, so wird dadurch eine systematische Tabellarisierung ebenfalls erschwert.

3.2.4 Selbstzerstörungskommandos

Der DS5002FP verfügt softwaremäßig über keine Möglichkeit, seinen aktuellen Schlüssel K zu überschreiben. Dies kann nur von außen durch den `SDI`-Pin, einen Batteriespannungsabfall oder durch das Unlock-Kommando der Firmware erreicht werden. Wenn die Anwendungssoftware sich selbst vernichten will, kann sie allerhöchstens versuchen, wichtige Daten im externen SRAM zu überschreiben, aber dies kann der Angreifer jederzeit unterbinden, da er die Kontrolle über die `R/`$\overline{\texttt{W}}$-Leitung hat. Auf das Vektor-RAM besteht für die Anwendungssoftware ebenfalls kein Schreibzugriff. Da es kein spezielles Suizid-Kommando zum Löschen von K gibt, besteht bei der systematischen Suche nach einer Befehlskombination in den Testläufen auch nicht die Gefahr, diesen Selbstzerstörungsmechanismus versehentlich auszulösen.

Ein Schutz gegen einen Angriff mit systematischer Befehlscodesuche kann daher darin bestehen, zusätzlich ein Selbstzerstörungskommando in den Befehlssatz mit aufzunehmen. Der Befehlscode dieses Suizidkommandos muss kurz genug sein, um mit großer Wahrscheinlichkeit bei einem Suchlauf während eines Angriffs aufzutauchen.

Beim DS5002FP sind fast alle Befehlscodes bereits mit normalen Anweisungen belegt, nur alle mit `a5h` beginnenden Maschinencodes sind noch nicht benutzt. Die Spezialregister sind im Registeradressraum im Bereich `80h` bis `ffh` angesiedelt, jedoch werden beim DS5002FP von 128 maximal möglichen Spezialregistern nur 29 Adressen genutzt. Jeder direktadressierte Schreib- oder Lesezugriff auf die verbleibenden 99 ungenutzen Spezialregisteradressen könnte als Selbstzerstörungskommando ausgewertet werden.

Bei Prozessorabstürzen durch vorübergehende besondere Umwelteinflüsse könnten jedoch die Selbstzerstörungskommandos auch ausgelöst werden, ohne dass ein Befehlssuch-Angriff durchgeführt wurde. Entweder wird ein vom Prozessor aus dem Speicher geladenes Byte zufällig so verändert, dass aus einer normalen Anweisung ein Selbstzerstörungsbefehl wird, oder in Folge eines Absturzes springt die außer Kontrolle geratene Befehlsführung zu Adressen, an denen nur Daten stehen, die auch zufällig Selbstzerstörungsbytefolgen enthalten könnten.

Wie schon die Registerfalle aus Abschnitt 3.1.3, so erhöhen auch die Selbstzerstörungsbefehle die Sicherheit auf Kosten der Zuverlässigkeit. Abhilfe kann jedoch ein Mechanismus schaffen, der einen einzeln auftretenden Selbstzerstörungsbefehl ignoriert, und den Schlüssel nur bei zeitlich gehäuftem Auftreten verschiedener dieser Befehle löscht. Es wäre denkbar n verschiedene Suizidbefehle zu implementieren und mit jedem dieser Befehle ein Monoflop zu assoziieren, das vom jeweiligen Befehl für k Minuten aktiviert wird. Sobald mehr als m der Monoflops gleichzeitig aktiviert wurden wird der Schlüssel gelöscht. Um sicher zu gehn dürfte der Angreifer dann nicht mehr als m/k Befehle pro Minute ausführen, was den Angriff erheblich in die Länge ziehen kann. Beim ersten Testlauf werden beispielsweise jeweils fünf Bytes, also schlimmstenfalls drei mögliche Befehle ausgeführt. Bei

$k = 30$ und $m = 5$ könnte dann nur noch alle sechs Minuten ein Befehl, also alle achtzehn Minuten eine Testsequenz ausgeführt werden. Bei 2^{17} verschiedenen Testsequenzen dauert dann alleine der Testlauf T1 etwa viereinhalb Jahre.

Im Gegensatz zur unkontrollierten Befehlsausführung bei einem Prozessorabsturz, die ohnehin nach wenigen Millisekunden durch den Watchdog abgebrochen werden sollte, werden bei einer Suche nach Befehlscodes systematisch alle Bytekombinationen durchgetestet. Es ist daher zu erwarten, dass beim Testlauf eines RAM-Emulationsangriffs fast alle Selbstzerstörungsbefehle in kurzer Zeit auftauchen, während bei Abstürzen nur einige wenige Bytes zufällig ausgeführt werden, bis der Prozessor wieder auf normale Befehlsfolgen stößt.

Unter den Spezialregistern befinden sich auch die Register `80h`, `90h`, `a0h` und `b0h` der Parallelschnittstellen des Prozessors. Wenn als Selbstzerstörungsbefehle ausschließlich Zugriffe auf bislang ungenutzte Spezialregister dienen, so hat ein geschickter Angreifer erstaunlicherweise immer noch eine Möglichkeit, eine gefahrlose Befehlssuche durchzuführen. Dazu beobachtet er den Prozessor und sucht eine Adresse a_0, bei der sich immer vier Buszugriffe nach einem Zugriff auf a_0 mehrere Bits einer Parallelschnittstelle gleichzeitig geändert haben. Dies deutet darauf hin, dass ab a_0 einer der Befehle beginnt, die mehrere Bits auf den Ports gleichzeitig ändern können. All diese Befehle haben in ihrem Befehlsformat im zweiten Byte die Adresse des beschriebenen Registers stehen (beispielsweise `a0h` bei Port P2). Da wir dieses zweite Befehlsbyte an der Adresse $a_0 + 1$ kennen, können wir sämtliche ersten Bytes durchprobieren, ohne befürchten zu müssen, ein anderes Register als das der Parallelschnittstelle mit Zugriffen zu treffen. Wenn wir nach dem ersten Testlauf die gesuchte `MOV`-Anweisung eine Adresse weiterschieben, so können wir das zweite Befehlsbyte anhand der bereits bekannten Verschlüsselungstabelle für diese Adresse festlegen und müssen keine gefährlich Suche mehr für diesen Wert durchführen. Für den Scheinzugriff der gesuchten `NOP`-ähnlichen Anweisung können wir statt $Z/8$ bis zum ersten Hinweis auf eine Bijektion einen festen Wert benutzen, so dass auch von dieser Stelle aus kaum möglicherweise gefährliche Zugriffe auf viele verschiedene Spezialregister ausgehen können. Dieses Beispiel zeigt, wie sorgfältig die Entwickler von Gegenmaßnahmen vorgehen müssen, um nicht alternative Vorgehensweisen des Angreifers (hier das Ausnutzen einer bereits vorhandenen Schreibanweisung auf die Parallelschnittstelle) zu übersehen.

3.2.5 Resetbeschränkungen

Mehrere hundert Resets pro Sekunde sind eine Situation, die im normalen Betrieb eines Prozessors höchst ungewöhnlich ist, sich aber bei einem Befehlssuchangriff kaum vermeiden lässt. Eine Schaltung, die innerhalb von m Minuten maximal n Resets zulässt und anschließend entweder den Schlüssel K löscht, oder wenigstens mit einem Monoflop die weitere Nutzung des Prozessors für k Minuten blockiert, ist daher eine wirkungsvolle Maßnahme, um Befehlssuchangriffe unpraktikabel zu machen. Eine derartige Schaltung könnte aus einem Kondensator bestehen, der bei jedem Reset mit einer bestimmten Ladungsmenge aufgefüllt wird und sich über einen Widerstand nur langsam wieder entladen kann. Überschreitet die Spannung des Kondensators aufgrund einer schnellen Abfolge von Resets eine Schwelle, dann wird der Schlüssel gelöscht oder der Prozessor wartet bis der Kondensator wieder entladen ist.

Neben Resets können auch Interrupts beim Befehlssuchangriff benutzt werden, um nach der Ausführung von zufälligen Befehlen den Prozessor für die nächste Testsequenz wieder in einen definierten Zustand zu bringen.

4 Busverschlüsselung in Arbeitsplatzrechnern

Bislang haben wir nur die Anwendung des Busverschlüsselungskonzepts an einem Microcontroller betrachtet. Microcontroller beherbergen in der Regel nur eine einzige Anwendungssoftware und oft nicht einmal ein von der Anwendung getrenntes Betriebssystem. Busverschlüsselungsprozessoren könnten prinzipiell auch in manipulationssicheren Arbeitsplatzrechnern oder Servern mit Multitasking-Betriebssystem und Mehrbenutzer-Betrieb sinnvoll eingesetzt werden.

Dazu ist es notwendig, nicht mehr einen einzelnen Schlüssel K für den gesamten Adressraum zu nutzen, sondern für einzelne Prozesse oder einzelne Speichersegmente getrennte Schlüssel zu verwenden. Die Busverschlüsselungslogik muss daher eng mit der Speicherverwaltungslogik des Prozessors verknüpft sein.

In diesem Kapitel wird der hypothetische Sicherheitsmikroprozessor mit Busverschlüsselung „*TrustNo 1*“ vorgestellt. Er ist für Sicherheitsanwendungen konzipiert, in denen weder dem Betriebssystem getraut werden darf, da es Sicherheitslücken aufweisen kann und vom Angreifer modifiziert worden sein könnte, noch der Systemhardware außerhalb des Prozessors, da sie ebenfalls zum Beispiel für einen RAM-Emulationsangriff oder für vergleichbare Eingriffe manipuliert worden sein könnte. Mit dem hier vorgestellten Konzept kann ein hochsicherer Kopierschutz für wertvolle Software erzielt werden. Ebenso kann sicherheitsrelevante Software gegen Manipulationen geschützt werden, obwohl der Angreifer vollen Zugriff zur Hardware außer dem Mikroprozessorchip selbst haben darf.

4.1 Hardware-Voraussetzungen

Als Ausgangsbasis für den Entwurf des *TrustNo 1* bietet sich ein Prozessor wie der Intel i386 an, bei dem der virtuelle Speicher nicht nur in Seiten, sondern auch in Segmente unterteilt werden kann. Eine Adresse besteht dann aus einem Segmentselektor und einem Offset. Der Segmentselektor identifiziert den Segmentdeskriptor, also einen Tabelleneintrag in dem das angesprochene Segment mit Angaben wie Basisadresse, Länge und ob dieses Segment beschreibbar oder ausführbar sein soll beschrieben ist. Bei einem Zugriff auf eine Adresse holt sich der Prozessor aus dem vom Segmentselektor identifizierten Segmentdeskriptor die Basisadresse, vergleicht den Offset mit der Segmentlänge, prüft die Zugriffsrechte und addiert die Basisadresse zum Offset, um die sogenannte lineare Adresse zu erhalten, die anschließend von der Seiten-Kachel-Tabelle in eine physikalische Adresse umgesetzt wird.

In unserem hypothetischen Prozessor befindet sich in einem geschützen Speicher eine Schlüsseltabelle, in der n Segmentschlüssel K_i $(0 < i \leq n)$ gespeichert werden können. Eine Schlüsseltabellengröße von $n = 255$ dürfte normalerweise ausreichend sein. Diese Schlüsseltabelle kann nur von einer in den Prozessor integrierten in einem manipulationssicheren Speicher untergebrachen Firmware beschrieben werden. Auch der Betriebssystemkern hat keine Möglichkeit, diese Schlüsseltabelle direkt auszulesen oder zu beschreiben, er kann aber Firmwareroutinen aufrufen, die diese Aufgaben nach unten beschriebenen Regeln übernehmen. Die Segmentdeskriptoren werden um ein Feld erweitert, das den Index i eines Schlüssels K_i aus der Schlüsseltabelle aufnehmen kann. Der spezielle Schlüsselindexwert $i = 0$ besagt, dass dieses Segment nicht verschlüsselt ist.

Die Speicherlogik kann eine Cachezeile wahlweise verschlüsselt oder unverschlüsselt in den Cache laden. Wir benutzen einen *write-back* Cache, dass heißt, jeder Schreibzugriff wird zunächst nur im Cache ausgeführt und erst später, wenn die entsprechende Cachezeile verdrängt wird, schreibt die Cachesteuerung die Daten wieder in den Hauptspeicher zurück. Mit jeder Zeile des Caches ist ein Schlüsselindex i assoziiert, in dem vermerkt wird, ob ($i \neq 0$) und mit welchem Schlüssel K_i diese Zeile beim Laden entschlüsselt wurde und daher vor dem Zurückschreiben in den Hauptspeicher wieder verschlüsselt werden muss. Wenn der Prozessor auf eine Adresse zugreift und diese bereits im Cache liegt, so überprüft er zuerst, ob der Schlüsselindex im Segmentdeskriptor mit dem der Cachezeile übereinstimmt. Ist dies der Fall, so benutzt der Prozessor sofort die Daten aus dem Cache. Stimmen diese beiden Indices nicht überein, so wird die Cachezeile verdrängt und mit passender Entschlüsselung neu aus dem Hauptspeicher geladen. War die angesprochene Adresse nicht im Cache vorhanden, so wird die Zeile in den Cache geladen und dabei je nach Wert des Schlüsselindexfeldes i im Segmentdeskriptor mit dem Schlüssel K_i entschlüsselt. Da nicht zu erwarten ist, dass auf die gleiche Adresse häufig abwechselnd mit verschiedenen Schlüsseln zugegriffen wird, ist es akzeptabel, wenn solche Zugriffe jeweils die entsprechende falsch entschlüsselte Cachezeile aus dem Cache verdrängen.

Die Speicherverwaltungslogik des Prozessors muss sicherstellen, dass auf alle Segmente, die mit einem Schlüssel K_i verschlüsselt sind, nur Maschinenbefehle Schreib-, Lese- oder Ausführungszugriff haben, die aus ebenfalls mit K_i verschlüsselten Codesegmenten heraus ausgeführt wurden. So wird verhindert, dass das Betriebssystem ein verschlüsseltes Programm entschlüsselt lesen oder modifizieren kann, oder dessen Funktionen unkontrolliert aufruft. Der Einsprung in ein verschlüsseltes Codesegment darf aus anders verschlüsselten Segmenten heraus nur an genau festgelegten Punkten erfolgen, zum Beispiel nur an der Adresse 0. Dies stellt sicher, dass das verschlüsselte Programm bestimmen kann, unter welchen Umständen einzelne Teile des Programms ausgeführt werden. Die Firmware des Prozessors ist ebenfalls verschlüsselt und kann damit vom Betriebssystem zwar aufgerufen, aber nicht ausgespäht werden.

4.2 Sichere Kontextwechsel

Wenn während der Ausführung von Befehlen, die aus einem verschlüsselten Codesegment stammen, ein Interrupt auftritt, so wird automatisch der gesamte den aktuellen Aktivitätsträger (*thread*) betreffende Zustand Z des Prozessors inklusive Programmzähler und Registerinhalte in einem für das Betriebssystem nicht zugänglichen Zwischenspeicher T abgelegt und anschließend werden die Register initialisiert. Damit kann nach der Interruptbehandlung die Arbeit des Prozessors im verschlüsselten Codesegment sofort wieder aufgenommen werden, ohne dass das Betriebssystem in der Interruptroutine die Möglichkeit hat, Einblick in die Registerinhalte der verschlüsselt ausgeführten Anwendung zu erhalten. Die Interrupt-Bearbeitung wird beendet, indem der in T gespeicherte Prozessorzustand Z wieder hergestellt wird. Der Zwischenspeicher T ist ein kleiner LIFO-Speicher, damit Interrupts höherer Priorität auch verschlüsselte Interrupt-Bearbeitungsroutinen niedrigerer Priorität unterbrechen können.

In einem Betriebssystem mit einem *preemptive scheduler* kann es passieren, dass nach einem Interrupt mit der Ausführung eines anderen Aktivitätsträgers fortgefahren werden soll. Es muss also eine Möglichkeit bestehen, den Inhalt von T in einer Tabelle des

Betriebssystems abspeichern zu können, ohne dass der Inhalt bekannt wird oder manipuliert werden kann. Daher verfügt der hier skizzierte Prozessor über eine spezielle Anweisung `SAVE_STATE`, die den im Stapel T zuoberst gespeicherten Zustand Z verschlüsselt im Hauptspeicher an einer als Parameter angegebenen Adresse ablegt. Ein entsprechendes zweites Kommando `RESTORE_STATE` entschlüsselt einen an der angegebenen Stelle im Hauptspeicher gesicherten Prozessorzustand und aktiviert diesen wieder. Damit kann eine Interruptroutine einen Kontextwechsel durchführen, ohne dabei Zugriff auf die Registerinhalte eines geschützten Anwendungsprogramms zu bekommen.

Wir benötigen einen Mechanismus, der sicherstellt, dass ein so gesicherter Zustand nur ein einziges mal wieder reaktiviert werden kann. Sonst könnte das Betriebssystem ohne Einverständnis der geschützten Software durch Mehrfachreaktivierung zwischengespeicherter Prozessorzustände unzulässige Schleifen ausführen oder Aktivitätsträger vervielfachen. Damit könnte ein Angreifer beispielsweise Softwarelizenzbedingungen unterlaufen, die nur eine maximale Anzahl von Funktionsausführungen zulassen.

Als Gegenmaßnahme besitzt der Prozessor auf dem Chip eine Tabelle mit einem Eintrag H_j für jeden geschützten Aktivitätsträger j. Auch diese Tabelle kann vom Betriebssystem nicht direkt ausgelesen oder beschrieben werden. Dem `SAVE_STATE`-Kommando wird vom Betriebssystem als Parameter der Index j des aktuellen Aktivitätsträgers mitgegeben. Das `SAVE_STATE`-Kommando erzeugt eine (beispielsweise 64-bit große) Zufallszahl und speichert diese in H_j ab. Dann fügt es dem obersten Prozessorzustand Z aus T etwas redundante Information hinzu (beispielsweise eine Prüfsumme oder nur einige Null-Bytes), verschlüsselt beides zusammen mit dem Schlüssel H_j und legt es im Hauptspeicher an der angegebenen Adresse ab. Das `RESTORE_STATE`-Kommando erhält ebenfalls als Parameter die Adresse des verschlüsselten Prozessorzustandes im Hauptspeicher sowie den Index j des entsprechenden Aktivitätsträgers. Es entschlüsselt die Zustandsdaten aus dem Hauptspeicher, löscht H_j, und überprüft anschließend die redundante mitentschlüsselte Information. Ist sie korrekt, so wird der gespeicherte Prozessorzustand wiederhergestellt und der Aktivitätsträger kann seine Tätigkeit wieder aufnehmen. Da durch `RESTORE_STATE` H_j gelöscht wird, kann kein geschützter Aktivitätsträger durch ein zweites `RESTORE_STATE` verdoppelt werden, da ohne das H_j der Zustand nicht mehr korrekt entschlüsselt wird und dies der Prozessor anhand der redundant mitverschlüsselten Information sofort feststellt.

Damit die Anwendung selbst neue Aktivitätsträger erzeugen kann, muss sie mit einem Kommando `TRANSFER_STATE` ähnlich einem Interrupt den aktuellen Prozessorzustand nach T kopieren, ohne dass er dabei wie bei einem Interrupt gelöscht wird. Diesen kann sich dann das Betriebssystem unter einem noch freien neuen Aktivitätsträgerindex j abspeichern und hat damit einen getrennten neuen Systemzustand. Die `TRANSFER_STATE`-Anweisung setzt für den alten Aktivitätsträger ein Statusbit, das im gespeicherten Zustand gelöscht wurde. Damit kann die Software den alten vom neuen Aktivitätsträger unterscheiden.

Für Aufrufe von Betriebssystemfunktionen gibt es einen speziellen `SUPERVISOR_CALL`-Maschinenbefehl, der zum einen wie bei einem normalen Betriebssystem in den Kontext des Systemkerns wechselt, der aber zusätzlich ähnlich wie bei einem Interrupt den Systemzustand verbirgt. Das Löschen der Registerinhalte, das bei Interrupts automatisch erfolgt, muss jedoch die Anwendungssoftware vor einer `SUPERVISOR_CALL`-Anweisung selbst übernehmen, da ja einige Registerinhalte als Parameter absichtlich an den Kern übergeben werden. Der Datentransfer größerer Parameter, die nicht in die Register passen, wird über unverschlüsselte Datensegmente vorgenommen.

4.3 Schlüsselmanagement

Nehmen wir an, das Sicherheitsziel besteht darin, Software gegen den nicht-lizenzierten gleichzeitigen Betrieb auf vielen Prozessoren zu schützen. Der Softwarehersteller V verkauft für die Software S eine Benutzungslizenz für einen bestimmten Prozessor P. Er muss dazu diese Software mit einem Schlüssel K_S verschlüsseln und dem Prozessor K_S so mitteilen, dass niemand sonst den Schlüssel K_S ermitteln oder in einen anderen Prozessor laden kann.

Zu diesem Zweck bieten sich asymmetrische Kryptosysteme an, also Verschlüsselungsverfahren, bei denen Schlüssel immer als Paar (K, K^{-1}) erzeugt werden, so dass Daten, die mit K verschlüsselt wurden, nur mit K^{-1} wieder entschlüsselt werden können und Daten die mit K^{-1} verschlüsselt wurden, wiederum nur mit K zu entziffern sind. Die Verfahren werden so entworfen, dass es ein extrem schwieriges Problem ist, zu bekanntem K den passenden Schlüssel K^{-1} zu finden und umgekehrt. Asymmetrische Kryptosysteme werden auch *public-key*-Kryptosysteme genannt, da sie es ermöglichen, einen der beiden Schlüssel öffentlich bekannt zu geben, während der zweite Schlüssel vom Eigentümer geheimgehalten wird. Auf diese Weise kann jeder mit dem öffentlichen Schlüssel eine Nachricht an den Schlüsseleigentümer schicken, die nur dieser mit seinem geheimen Schlüssel lesen kann. Dies erspart den aufwendigen Austausch von gegenseitigen Geheimschlüsseln, die nur jeweils zwei Kommunikationspartner kennen dürfen. Entsprechend können auch digitale Unterschriften vorgenommen werden: Wenn der Eigentümer eines Schlüsselpaares eine Nachricht mit seinem Geheimschlüssel verschlüsselt, dann kann jeder diese mit dem entsprechenden allgemein bekannten öffentlichen Schlüssel entschlüsseln. Ist dies erfolgreich möglich, so kann die Nachricht nur vom Besitzer des entsprechenden geheimen Schlüssels verschlüsselt worden sein, die Nachricht muss also von ihm stammen. Gängige asymmetrische Kryptosysteme sind das RSA- und das ElGamal-Verfahren, die zur Verschlüsselung eine Nachricht als Element eines endlichen Körpers darstellen und die Nachricht mit dem Schlüssel potenzieren. Angriffe auf diese Verfahren sind ähnlich schwierig wie das Produkt aus zwei 200-stelligen Primzahlen wieder in seine Faktoren zu zerlegen oder den diskreten Logarithmus in einem sehr großen endlichen Körper zu berechnen. Gute symmetrische Kryptosysteme wie etwa DES, bei denen der gleiche Schlüssel zum Ver- und Entschlüsseln eingesetzt wird, können sehr effizient implementiert werden, denn um hinreichende Sicherheit zu erreichen sind nur etwa 64–128 Bit Schlüsselgröße notwendig, da die praktikabelsten Angriffe aus dem systematischen Ausprobieren aller Schlüssel bestehen. Bislang bekannte asymmetrische Kryptosysteme dagegen erfordern einen wesentlich größeren Rechenaufwand und Schlüssellängen im Bereich von etwa 1000 Bit, da ausgefeilte Angriffe bekannt sind, die wesentlich effizienter arbeiten als das systematische Durchsuchen aller Schlüssel. Aus diesem Grund werden asymmetrische Kryptosysteme nur eingesetzt, um symmetrische Schlüssel zu verschicken, mit denen dann die eigentlichen umfangreichen Daten verschlüsselt werden [Sch96]. Im Folgenden werden wir das Ver- beziehungsweise Entschlüsselungsergebnis einer Nachricht M mit einem Schlüssel K wie in der Literatur über Kryptoprotokolle üblich mit $\{M\}_K$ bezeichnen.

Ausgangspunkt für das folgende Konzept ist, dass der Chiphersteller C des *TrustNo 1* Prozessors vertrauenswürdig ist, und nicht versuchen wird, von Softwareherstellern vertriebene verschlüsselte Software zu entschlüsseln, was er technisch leicht könnte. Der Chiphersteller erzeugt ein Schlüsselpaar (K_C, K_C^{-1}) und macht K_C bei allen Softwareherstel-

lern bekannt. Jeder Softwarehersteller kann für sich entscheiden, welchem Hersteller C er vertraut.

Für jeden unserer hypothetischen Sicherheitsprozessoren von Typ *TrustNo 1* wurde bei der Herstellung ein asymmetrisches Schlüsselpaar (K_P, K_P^{-1}) sowie ein symmetrischer Schlüssel K'_P erzeugt und im Prozessor abgespeichert. Jeder Prozessor enthält individuelle Schlüssel K_P, K_P^{-1} und K'_P. Der Schlüssel K_P ist öffentlich bekannt, d.h. das Betriebssystem kann ihn vom Prozessor erfragen und der Hersteller kann ihn auch als eine Art Seriennummer auf das Gehäuse aufdrucken. Dagegen sind K_P^{-1} und K'_P im manipulationssicheren Speicher des Prozessors untergebracht und nicht einmal für das Betriebssystem sondern nur für die Prozessorfirmware zugänglich. Neben dem öffentlichen Schlüssel K_P kann vom Prozessor auch noch eine digitale Unterschrift $\{K_P\}_{K_C^{-1}}$ abgefragt werden, die jedem Softwarehersteller beweist, dass K_P wirklich vom Originalhersteller des *TrustNo 1* Prozessors erzeugt wurde und nicht von einem Angreifer.

Der Hersteller V erzeugt für sein Softwareprodukt S einen Schlüssel K_S für die symmetrische Busverschlüsselung, und macht $\{S\}_{K_S}$ allgemein zugänglich, also zum Beispiel auf kostenlosen CD-ROMs, Internet-Servern, Telefon-Mailboxsystemen, etc. Da S verschlüsselt wurde, kann niemand ohne Weiteres etwas damit anfangen. Wenn der Besitzer des Prozessors P für diesen eine Lizenz für den Gebrauch von S erwerben möchte, dann teilt er dem Hersteller von S den öffentlichen Schlüssel K_P seines Prozessors sowie die Unterschrift $\{K_P\}_{K_C^{-1}}$ des Chipherstellers C dazu mit und bezahlt die Software. Daraufhin erhält er $\{K_S\}_{K_P}$ zugesandt.

Dem Betriebsystem, das auf Prozessor P abläuft, stehen nun die verschlüsselte Software $\{S\}_{K_S}$ sowie der gekaufte Schlüssel $\{K_S\}_{K_P}$ zur Verfügung. Um die Software benutzen zu können, übergibt das Betriebssystem an die manipulationsgeschützte Firmware das Datenpaket $\{K_S\}_{K_P}$, womit die Firmware $\{\{K_S\}_{K_P}\}_{K_P^{-1}} = K_S$ berechnet. Damit diese Berechnung, die wegen des Rechenaufwandes für asymmetrische Kryptosysteme einige Sekunden dauern kann, nicht bei jedem Neustart von S durchgeführt werden muss, übergibt die Firmware an das Betriebssystem anschließend das Datenpaket $\{K_S\}_{K'_P}$, das vom Betriebssystem zusammen mit der ausgelieferten Software $\{S\}_{K_S}$ auf dem Massenspeicher abgespeichert wird.

Die verschlüsselte Software $\{S\}_{K_S}$ besteht in einer realen Implementation nicht aus einem großen Datenblock, der komplett mit K_S verschlüsselt wurde, sondern sie wird in einem Dateiformat ausgeliefert, das in einem unverschlüsselten Dateikopf Informationen über die Anzahl und Länge der von S benötigten Segmente enthält sowie den pro Segment einzeln mit K_S verschlüsselten Inhalt dieser Segmente, soweit sie vorbelegt werden müssen. Wenn die Software gestartet wird richtet das Betriebssystem nach den Angaben im unverschlüsselten Dateikopf von S zunächst unverschlüsselte neue Code-, Daten-, Stack- und andere Segmente ein und läd in sie die entsprechenden verschlüsselten Komponenten aus $\{S\}_{K_S}$. Anschließend wählt das Betriebssystem einen noch freien Schlüsselindex i und übergibt diesen zusammen mit $\{K_S\}_{K'_P}$ an die Firmware, die daraufhin K_S entschlüsselt und als K_i in der Schlüsseltabelle des Prozessors abspeichert. Nun trägt das Betriebssystem in die Segmentdeskriptoren von S, die laut Dateikopf verschlüsselt sein müssen, den Schlüsselindex i ein. Ab diesem Zeitpunkt sieht der Prozessor die Software S mit ihren Daten unverschlüsselt, aber nur solange Befehle ausgeführt werden, die in einem der verschlüsselten Segmente von S stehen.

Bevor das Betriebssystem die Software starten kann, muss es sich noch einen Tabellenein-

trag für den neuen Aktivitätsträger anlegen. In der Aktivitätsträgertabelle des Prozessors muss ein freier Eintrag H_j gefunden werden, in dem der Schlüssel für den Prozessorzustand gelagert werden kann. In der Aktivitätsträgertabelle des Betriebssystems wird neben den bei anderen Betriebssystemen üblichen Daten auch j abgespeichert sowie ein Platz für den verschlüsselten Prozessorzustand reserviert.

Wenn all diese Vorbereitungen getroffen wurden, wird die Software S mit entsprechenden Registerwerten über die noch erlaubte Einsprungstelle im Codesegment aufgerufen.

Es ist auch ein Mechanismus vorstellbar, der es ermöglicht, Softwarelizenzen zurückzugeben, oder auf einen anderen Prozessor umschreiben zu lassen. Dazu existiert im Prozessor P eine „schwarze Liste“ in die der Besitzer von P jedes Programm S, das er auf P nicht mehr weiter benutzen will, von der Firmware eintragen lassen kann. Dazu übergibt er der Firmware den vom Softwarehersteller V ursprünglich gekauften Schlüssel $\{K_S\}_{K_P}$. Die Firmware speichert daraufhin in der schwarzen Liste $\{K_S\}_{K'_P}$ ab und löscht jeden eventuell noch in der Schlüsseltabelle vorhandenen Eintrag $K_i = K_S$.

Jedesmal wenn die Firmware ein Datenpaket $\{K_S\}_{K'_P}$ erhält, um dies in K_i abzulegen, überprüft sie, ob dieses Datenpaket nicht schon auf der schwarzen Liste steht und verhindert das Entschlüsseln von $\{K_S\}_{K'_P}$ in diesem Fall. Die schwarze Liste ist ein nur einmal beschreibbarer Speicher und einmal eingetragene Datensätze können nie mehr entfernt werden, ohne den Prozessor zu beschädigen. Damit besteht für den Benutzer des Prozessors P keine Möglichkeit mehr, die Software S jemals wieder auf diesem Prozessor auszuführen. Er hat damit die Softwarelizenz zurückgegeben. Er kann sich von der Firmware des Prozessors eine Bescheinigung $B = \{h(K_S)\}_{K_P^{-1}}$ ausstellen lassen, und diese zusammen mit K_P an den Hersteller V schicken. Dabei ist h eine kryptografisch sichere Einweg-Hashfunktion. Das Verschlüsseln mit dem nicht-öffentlichen Schlüssel K_P^{-1} entspricht einer digitalen Unterschrift des Prozessors P unter die Aussage, dass er künftig keine Software mehr mit dem Schlüssel K_S entschlüsseln wird. Der Besitzer des Prozessors kann mangels Kenntnis von K_P^{-1} diese Bescheinigung nicht fälschen. Der Hersteller hat in seiner Kundendatenbank einen Lizenzvertrag abgespeichert, der den Prozessor, für den die Software-Lizenz vergeben wurde, mit K_P identifiziert. Er kann daher $\{B\}_{K_P}$ berechnen und wenn $\{B\}_{K_P} = h(K_S)$ ist, so weiß er, dass die Rückgabe der Softwarelizenz vollzogen wurde. Er kann nun beispielsweise aufhören, monatliche Lizenzgebühren zu verlangen, oder er kann kostenlos eine neue Lizenz $\{K_S\}_{K_Q}$ für einen neuen Prozessor Q ausstellen, wenn der Kunde sich einen neuen Rechner gekauft hat und die Software nun statt auf P auf dem neuen Prozessor Q laufen lassen möchte.

Wenn ein Kunde einmal $\{K_S\}_{K'_P}$ aus Versehen in die schwarze Liste seines Prozessors eingetragen hat, dann reicht es nicht, eine neue Lizenz $\{K_S\}_{K_P}$ vom Hersteller zu besorgen, da dies ja wieder genau der gleiche Datensatz wie bei der ersten jetzt ungültigen Lizenz wäre. Statt dessen müsste die gesamte Software vom Hersteller mit einem neuen Schlüssel K'_S verschlüsselt und ausgeliefert werden. Um diesen Aufwand zu vermeiden, kann bei einer realen Implementation noch eine Schlüsselebene zwischen K_S und K_P eingeschoben werden, so dass der Prozessor mit K_P^{-1} einen symmetrischen Lizenzschlüssel K_L entschlüsselt, mit dem dann schließlich erst K_S entschlüsselt wird. Dann wird nur noch $\{K_L\}_{K'_P}$ in die schwarze Liste eingetragen, und K_L wird in der nächsten Lizenz anders sein, ohne dass die gesamte Software auf CD-ROM extra neu verschlüsselt werden muss.

Sollte einmal ein Prozessor P defekt werden bevor er Rückgabebescheinigungen für die

für ihn lizenzierten Programme ausstellen konnte, so muss ihn der Besitzer an den Prozessorhersteller C einschicken. Dieser überprüft anhand des Gehäuseaufdrucks oder bei beschädigtem Gehäuse anhand eines auf den Chip individuell mit Laser eingebrannten Codes, ob der zerstörte *TrustNo 1* Prozessor wirklich den öffentlichen Schlüssel K_P benutzte. In diesem Fall stellt der Prozessorhersteller einen wiederum mit K_C^{-1} unterschriebenen „Totenschein“ für P aus, der bestätigt, dass der Prozessor mit dem öffentlichen Schlüssel K_P zerstört wurde. Diese Bescheinigung kann der Kunde dann an alle Hersteller der von ihm gekauften Software schicken und diese können die Unterschrift des Chipherstellers überprüfen und daraufhin dem Kunden kostenlos eine neue Lizenz für den Ersatzprozessor ausstellen.

4.4 Einsatzmöglichkeiten

Das hier vorgestellte Softwareschutz-Verfahren mag auf den ersten Blick recht umständlich erscheinen, da für den Erwerb der Software Datenpakete mit dem Hersteller ausgetauscht werden müssen. Allerdings wird derzeit ohnehin für das Internet und andere Online-Dienste eine Infrastruktur für elektronischen Verkauf an Endverbraucher aufgebaut, wobei die Bezahlvorgänge mit kryptografischen Protokollen abgewickelt werden. Daher entsteht kein nennenswerter Zusatzaufwand, wenn auch der Lizenzdatenaustausch über die gleichen Kanäle wie der Bezahlvorgang abgewickelt wird.

Ob sich ein Sicherheitsprozessor wie der hier skizzierte durchsetzen kann, hängt nicht nur von technischen Faktoren ab. Bislang haben Hardware-Hersteller davon profitiert, dass viele Software-Raubkopien für ihr Produkt verfügbar waren, da zumindest unter Privatanwendern die Verfügbarkeit von Raubkopien ein Grund sein kann, sich für eine bestimmte Systemarchitektur zu entscheiden. Daher ist es vorstellbar, dass Prozessorhersteller gar kein besonderes Interesse an einem Sicherheitsprozessor für einen umfassenden Kopierschutz haben. Hauptanwendung für einen Sicherheitsprozessor wie den geschilderten sind daher zunächst eher Arbeitsplatzrechner, auf denen sicherheitskritische Anwendungen mit geheimen Algorithmen und Daten benutzt werden.

5 Zusammenfassung

In den verschiedensten Anwendungsbereichen werden Schutzmechanismen gegen die unbefugte physikalische Manipulation von Computersystemen durch den regulären Besitzer oder durch Saboteure und Diebe benötigt. Im Gegensatz zum Entwurf sicherer Betriebssysteme und sicherer Kommunikationsprotokolle wurden physikalische Schutzmaßnahmen von Rechnern bislang in der frei zugänglichen wissenschaftlichen Literatur nur vereinzelt behandelt. Dies führte dazu, dass Anwendungsentwickler den von manipulationssicheren Systemen – wie etwa Chipkarten – gebotenen Widerstand gegen ernsthafte Angriffe oft erheblich überschätzen. In der Literatur existieren verschiedene Klassifikationen von Sicherheitsmechanismen für die Mächtigkeit eines potentiellen Angreifers, den Aufwand den ein erfolgreicher Angriff auf ein Sicherheitssystem erfordert und den Umfang der Schutzmaßnahmen. Existierende Sicherheitskonzepte lassen sich grob unterteilen in Einchip-Systeme, Module mit batteriegepuffertem SRAM und sicherem Gehäuse sowie Busverschlüsselungssysteme. Letztere stellen aufgrund der im Vergleich zu anderen Sicherheitskonzepten fehlenden engen Kapazitätsgrenzen ein sehr attraktives Schutzkonzept dar.

Der Microcontroller DS5002FP ist der derzeit verbreiteteste Mikroprozessor mit Busverschlüsselung und Batteriepufferung. Er galt bislang als der sicherste Mikroprozessor für Anwendungen, die mehr als wenige hundert Bytes RAM benötigen. Im Rahmen der vorliegenden Arbeit wurde ein neuer Angriff gegen Busverschlüsselungssysteme entwickelt. Er lässt sich einsetzen, wenn die Datenbusverschlüsselung eine kleine Blockgröße aufweist. Dieser Angriff umfasst die Emulation des Hauptspeichers eines Sicherheitssystems durch eine spezielle Schaltung sowie die systematische Suche nach den verschlüsselten Befehlscodes einzelner Maschinenbefehle durch Beobachten der Reaktion der CPU auf Testbytes (*cipher instruction search attack*). Die Entwicklung einer entsprechenden Ausleseschaltung samt Steuersoftware erlaubte mir, das Auslesen der geschützten Software aus DS5002FP-basierten Sicherheitssystemen praktisch unter realistischen Bedingungen zu demonstrieren.

Es existiert eine Reihe von möglichen Gegenmaßnahmen gegen diesen bisher bei der Entwicklung von Busverschlüsselungssystemen nicht berücksichtigten Angriff in Hard- und Software. Einige der hier vorgestellten Ideen, wie etwa die Selbstzerstörungskommandos oder die Resetbegrenzung, erlauben es, mit geringem Aufwand existierende Krypto-Prozessoren wie den DS5002FP in ihrer Sicherheit erheblich zu verbessern.

Künftige Sicherheitsprozessoren, welche Busverschlüsselung auch in modernen Arbeitsplatzrechnern mit Mehrprozessbetrieb einsetzen könnten, erfordern eine Reihe neuer Konzepte. Hier neu vorgestellte Mechanismen erlauben beispielsweise die Einbindung eines Cache-Speichers in das Busverschlüsselungssystem, die sichere Handhabung eines Kontextwechsels, den sicheren Schlüsseltransfer beim Kauf verschlüsselter Software und die sichere Übertragung einer Softwarelizenz auf einen neuen Prozessor. Prozessoren mit Busverschlüsselung erlauben einen sehr wirksamen Kopierschutz für Software zu realisieren. Sie könnten sich in Zukunft zu einem wesentlichen Bestandteil zahlreicher Computersicherheitskonzepte entwickeln.

Anhang: Implementation der Ausleseschaltung

Für eine praktische Demonstration des in Kapitel 2 vorgestellten Angriffs gegen den Busverschlüsselungsprozessor DS5002FP war es erforderlich, eine RAM-Emulator-Hardware zu entwickeln. Prinzipiell wäre es möglich gewesen, den Angriff fast ausschließlich in Software zu implementieren. Dazu hätten nur sämtliche Pins eines DS5002FP mit einem etwa 80-kanaligen parallelen Ein-/Ausgabe-Interface in einem PC verbunden werden müssen. Eine Simulations-Software im PC hätte dann sämtliche Signale der Peripherie einschließlich der Taktsignale erzeugt, hätte sie dem Prozessor zur Verfügung gestellt und hätte auf die Signale des Prozessors wie die SRAM-Bausteine eines echten Mikroprozessorsystems reagiert. Da zunächst aufgrund der vorhandenen Dokumentation [Dal93] noch nicht vollständig abschätzbar war, welche weiteren Funktionen vom RAM-Emulator im Lauf der Entwicklung des Ausleseverfahrens benötigt werden könnten, hätte die vollständige Software-RAM-Emulation die größtmögliche Flexibilität geboten. Allerdings muss der DS5002FP laut Datenblatt mindestens mit 1 MHz Taktfrequenz betrieben werden. Die reine Softwarelösung hätte auf einem PC diese Vorgabe nicht einhalten können und alle Versuche hätten außerhalb der spezifizierten Betriebsbedingungen stattfinden müssen.

Darüber hinaus war das Ziel der Untersuchung, einen Angriff auf den DS5002FP nicht nur an einem bereits vor Einspielen der geschützten Software voll in einen Versuchsaufbau integrierten Musterprozessor zu simulieren, sondern auch Angriffe unter realistischen Bedingungen durchführen zu können, um mehr über die dabei auftretenden praktischen Probleme zu erfahren. Bei einem echten Ausleseversuch liegt eine Systemplatine mit Prozessor und SRAM vor, wobei die Programm- und Schlüsseldaten nur von der Spannung einer kleinen Lithiumbatterie erhalten werden. Eine kurzzeitige Unterbrechung oder ein Kurzschluss der Batteriespannung hätte den unwiderruflichen Verlust der Daten zur Folge. Daher musste die Ausleseschaltung so geplant werden, dass die Eingriffe in die untersuchte Hardware auf das notwendige Minimum beschränkt werden. Ein vollständiges Auslöten des Prozessors bei anliegender Batteriespannung scheidet daher beispielsweise aus.

Der DS5002FP muss während des Angriffs völlig mit seinem System verbunden bleiben, nur die *chip-enable*- und *read/write*-Leitungen zu den SRAMs werden unterbrochen. Die entwickelte Schaltung ist in den Diagrammen am Ende dieses Abschnitts dargestellt. Sie weist folgende Eigenschaften auf:

- Der Prozessor kann wahlweise vollständig über die Ausleseschaltung oder über das untersuchte System mit Ressourcen wie Takt, Versorgungsspannung, Batteriespannung, SRAM, serielle Schnittstelle zum PC, usw. beliefert werden. Über Steckbrücken kann entschieden werden, ob die Schaltung mit einem aufgesteckten Prozessor auf einer Adapterplatine für Versuche oder mit einem aufgesteckten Testclip für realistische Angriffe genutzt werden soll. Insbesondere ist die Schaltung in der Lage, sich vollständig dem Takt aus dem untersuchten System anzupassen.
- Die Schaltung führt sämtliche Schritte während der Prozessor eine Testsequenz bearbeitet autonom durch. Der PC füllt nur den Befehls-FIFO, gibt den Reset frei, wartet darauf, dass der Befehls-FIFO entleert wurde und liest die Aufzeichnungs-FIFOs aus.
- Die Ausleseschaltung benötigt neben einem PC mit einer Schnittstellenkarte und entsprechender Steuersoftware keine weiteren Hilfsmittel.

- Sie kann als normales eigenständiges DS5002FP-Entwicklungssystem genutzt werden, wenn der PC durch eine 5 V Spannungsquelle ersetzt wird.
- Die ist flexibel und kann leicht erweitert werden.

Die Ausleseschaltung verfügt über einen 9-bit Datenbus `D[0..8]` und einen 17-bit Adressbus `A[0..16]`, auf welche der PC über 26 seiner Ein-/Ausgänge zugreifen kann. Die Datenbusanschlüsse des DS5002FP sind direkt an `D[0..7]` angeschlossen, da Messungen gezeigt haben, dass sie sich hochohmig verhalten, solange der Prozessor über seinen $\overline{\texttt{VRST}}$-Pin im Reset-Zustand gehalten wird. Die Adressbusanschlüsse des Prozessors dagegen werden niemals hochohmig. Deswegen mussten sie durch die beiden Bustreiber U9 und U10 von `A[0..14]` getrennt werden. Der PC kann dadurch über die Steuerleitung `ADEN` (*CPU address enable*) den Dallas-Prozessor vom Adressbus der Ausleseschaltung abkoppeln, um selbst Adressen anlegen und auf den 128 Kilobyte SRAM-Baustein U3 zugreifen zu können.

Die Ausleseschaltung besitzt insgesamt vier 9-bit breite FIFO-Speicher vom Typ Cypress CY7C433 mit einer Kapazität von je 4096 Worten. Während der DS5002FP läuft wird `ADEN` auf 5 V gehalten und so können die beiden FIFOs U4 und U5 die Ereignisse auf dem Adressbus mitprotokollieren. Das 9. Bit von U4 zeichnet den Wert des `R/`$\overline{\texttt{W}}$-Ausgangs des Prozessors auf. Der FIFO U6 zeichnet eine der vier parallelen Schnittstellen P0 bis P3 der CPU auf, je nachdem welche der vier Steckleisten JP15 bis JP18 mit acht Steckverbindern besetzt ist. Die beobachtete Schnittstelle wird durch *pull-up*-Widerstände unterstützt, da es sich um Ausgänge mit offenem Kollektor handelt.

Das Herz der Ausleseschaltung ist der programmierbare Logikbaustein (CPLD) Lattice ispLSI 1023, der die gesamte Ablaufsteuerung enthält. Dieser Baustein kann über JP8 und die Druckerschnittstelle eines PCs auch im bereits fertiggestellten System mit einer neuen Programmierung ausgestattet werden, was die Entwicklung des Systems sehr vereinfacht. Die drei Aufzeichnungs-FIFOs U4 bis U6 können vom PC mit den Steuerleitungen $\overline{\texttt{RADLF}}$ (*read address low FIFO*), $\overline{\texttt{RADHF}}$ (*read address high FIFO*) und $\overline{\texttt{RPOF}}$ (*read port FIFO*) direkt über den Datenbus `D[0..8]` ausgelesen werden. Mit der Steuerleitung $\overline{\texttt{RSTF}}$ (*reset FIFOs*) kann der PC alle vier FIFOs zurücksetzen und damit entleeren.

Der FIFO-Speicher U7 enthält die verschlüsselten Befehlscodes, die dem Prozessor während des Angriffs vorgespielt werden. Er kann vom PC direkt über die Steuerleitung `PC8` = $\overline{\texttt{WINSF}}$ gefüllt werden, die nur durch den Logikbaustein geleitet wurde, um in einer möglichen aber derzeit nicht benötigten Erweiterung den Befehls-FIFO auch zur Aufzeichnung des Datenbusses benutzen zu können, wozu neben dem PC auch die Steuerlogik in der Lage sein müsste, U7 beschreiben zu können. Das 9. Ausgangsbit des Befehls-FIFOs teilt der Steuerlogik das `DOFIFO`-Signal mit, das festlegt, ob tatsächlich die anderen 8 Bits aus dem FIFO an die CPU übergeben werden sollen, oder ob dieser Speicherzugriff an das SRAM weitergeleitet werden soll. Entsprechend kann die Steuerlogik dann über das $\overline{\texttt{GATE}}$ signal entscheiden, ob über den Bustreiber U13 der FIFO mit dem Datenbus verbunden wird oder ob die Leseleitung eines RAM-Bausteins aktiviert werden soll. Dies kann entweder über $\overline{\texttt{CSRAM}}$ (*chip select RAM*) der SRAM-Chip der Ausleseschaltung sein, oder über eine der vier Leitungen $\overline{\texttt{CES1}}$ bis $\overline{\texttt{CES4}}$ einer der bis zu vier unterstützten SRAM-Chips im untersuchten System, deren *chip-select*-Eingänge von den $\overline{\texttt{CE1}}$ bis $\overline{\texttt{CE4}}$ Ausgängen der CPU getrennt wurden. An Steckverbinder JP13 steht neben den *chip-select*-Signalen für die SRAMs des untersuchten Systems auch noch das `RWS`-Signal für die $\overline{\texttt{WE}}$-Eingänge (*write enable*) zur Verfügung.

Die Anschlüsse für die DS5002FP-CPU U1 sind so ausgeführt, dass sie für Versuchszwecke einen auf eine Adapterplatine aufgelöteten Prozessor aufnehmen können, aber auch alternativ über Flachbandkabel und einen SMD-Testclip den Anschluss einer CPU in einem untersuchten System zulassen. Entsprechend diesen beiden Betriebsarten lassen sich Stromversorgung, Batterieversorgung und der Schwingquarz über die Steckbrücken JP3, JP4, JP9 und JP10 abtrennen. Die Diode D2 schützt eine eventuell im untersuchten System vorhandene Lithium-Batterie vor versehentlicher Entladung. Die Batteriespannung wird wegen des vernachlässigbaren Stroms nur mit einem Spannungsteiler aus R1 und R2 erzeugt. Die Schottky-Diode D1 garantiert, dass die Versorgungsspannung niemals unter $-0,3$ V abfallen kann, was laut [Dal93] bei einem batteriegepufferten System einen gefährlichen *latch-up*-Effekt auslösen könnte. Mit den Steckbrücken JP6 und JP7 kann entschieden werden, ob der CPU-Reset über $\overline{\texttt{VRST}}$ oder `RST` ausgelöst werden soll. Es hat sich gezeigt, dass nur über $\overline{\texttt{VRST}}$ die Buszugriffzyklen der CPU unterbrochen werden, weshalb JP6 immer geschlossen bleiben sollte. Die grüne LED D4 signalisiert eine anliegende Versorgungsspannung, die rote LED D3 meldet, dass die CPU im Resetzustand gehalten wird.

Mit JP5 kann über den `MSEL`-Eingang dem Prozessor mitgeteilt werden, ob er an vier einzelne 32-Kilobyte-RAMs oder an einen 128-Kilobyte-Chip angeschlossen ist. Davon hängt ab, ob die obersten beiden Adressbits als decodiertes eins-aus-vier Signal über $\overline{\texttt{CE1}}$ bis $\overline{\texttt{CE4}}$ ausgegeben werden sollen, oder ob $\overline{\texttt{CE3}}$ und $\overline{\texttt{CE2}}$ dem 16. und 17. Adressbit entsprechen und nur $\overline{\texttt{CE1}}$ als *chip-select*-Signal genutzt wird. Der Pegel am `MSEL`-Pin wird ebenso wie $\overline{\texttt{CE1}}$ bis $\overline{\texttt{CE4}}$ direkt der Steuerlogik zugeführt, die daraus immer und unabhängig davon welche Speicherkonfiguration die CPU sieht ein internes einheitliches *chip-select*-Signal erzeugt sowie die Adressbusleitungssignale `A15` und `A16` rekonstruiert.

Über einen Pegelkonverter U2 wird die serielle Schnittstelle des Prozessors mit der seriellen EIA-232-E Schnittstelle des PCs verbunden. Über die `DTR`-Leitung des PCs kann der Firmwaremodus des Dallas-Prozessors mit seinen $\overline{\texttt{PROG}}$-Eingang aktiviert werden, so dass über die serielle Schnittstelle der PC die Firmwarekommandos ansprechen und Software in den Prozessor laden kann.

Der Steuerrechner ist ein einfacher IBM AT kompatibler PC mit einem Intel i386 Prozessor und 640 Kilobyte RAM. Er verfügt über eine sehr einfache Datenerfassungs-Einschubkarte mit drei Intel 8255 Parallel-Ein-/Ausgabe-Bausteinen mit je 24 Bits und einen Intel 8253 Zählerbaustein mit drei unabhängigen 16-bit Zählern. Aus der Sicht der Steuersoftware stellt das Auslesesystem eine Reihe von Steuerregistern zur Verfügung, die in Abbildung A.1 dargestellt sind. Diese Register der 8255-Bausteine erlauben den Zugriff auf die Busse des Auslesesystems, die Umschaltung der Zugriffsrichtung auf die Busse und den Zugriff auf die oben genannten FIFO-Steuerleitungen. Sie erlauben ferner über die `PC`n-Bits die Steuerlogik zu kontrollieren, über `PRST` abzufragen, ob der Prozessor noch läuft und über $\overline{\texttt{RECEF}}$ die Füllung der Aufzeichnungs-FIFOs zu prüfen. Die Anordnung der Steuerbits in den Registern ist im Wesentlichen dadurch bestimmt, dass bei einer Richtungsumschaltung für einen Ausgang eines 8255-Bausteins alle anderen Ausgänge ihren Wert ändern können. Daher werden von den 8255-Bausteinen 1 und 3 die umschaltbaren Busleitungen bedient, während Baustein 2 die Steuerleitungen mit fester Übertragungsrichtung realisiert.

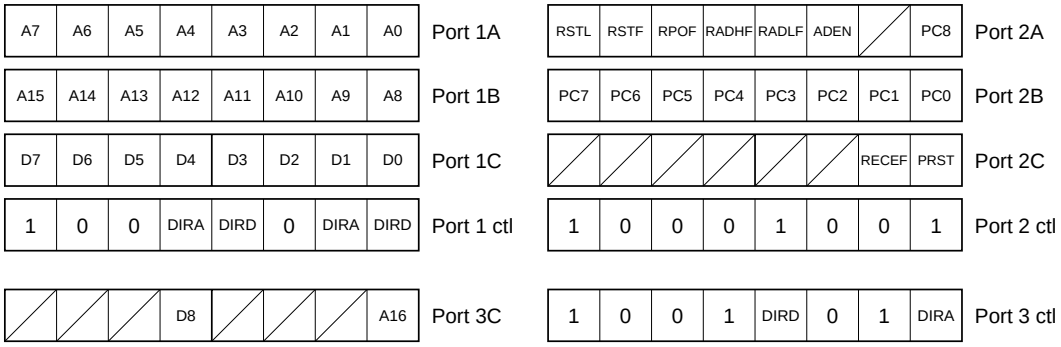


Abbildung A.1: Steuerregister der Ausleseschaltung aus der Sicht der PC-Software. `DIRA` und `DIRD` legen fest, ob der PC den Adress- bzw. Datenbus treibt (0) oder beobachtet (1).

Mit der Steuerleitung `PC0` kann entschieden werden, ob überhaupt eine Umschaltung auf den FIFO stattfinden soll und mit `PC1` wird festgelegt, ob der Prozessor den SRAM der Ausleseschaltung oder den SRAM im untersuchten System benutzen soll. Wenn der Prozessor ohne FIFO-Umschaltung das SRAM im untersuchten System benutzt, so kopiert sich die Ausleseschaltung bei jedem beobachteten Speicherzugriff den übertragenen Wert in das eigene SRAM. Mit dieser Funktion kann vor Eingriffen in das untersuchte System gefahrlos eine Sicherheitskopie großer Teile des SRAMs angefertigt werden. Mit `PC5` und `PC6` erhält der PC Lese- und Schreibzugriff auf das SRAM der Ausleseschaltung, um beispielsweise die Sicherheitskopie auslesen zu können. `PC7` kontrolliert die Resetleitung des Prozessors.

Über den Anschluss $\overline{\texttt{ARM}}$ kann ein Triggermechanismus aktiviert werden, der bei der nächsten Flanke auf dem `TRIG`-Eingang die Umschaltung auf den FIFO durchführt. Die Steuerlogik verfügt über mehrere Flipflops, mit denen sichergestellt wird, dass bei der Umschaltung vom SRAM- auf den FIFO-Betrieb alle Zeitanforderungen für die Steuersignale eingehalten werden. Die Taktsignale dieser Flipflops und anderer Ausgabesignale müssen in der richtigen Reihenfolge zueinander auftreten. Zu diesem Zweck werden mit einer asynchronen Logik aus den *chip-enable*-Signalen, weitere zeitverzögerte Taktsignale erzeugt (Abbildung A.2). Aus diesen werden dann die anderen Signale mit den erforderlichen zeitlichen Abständen der Flanken erzeugt.

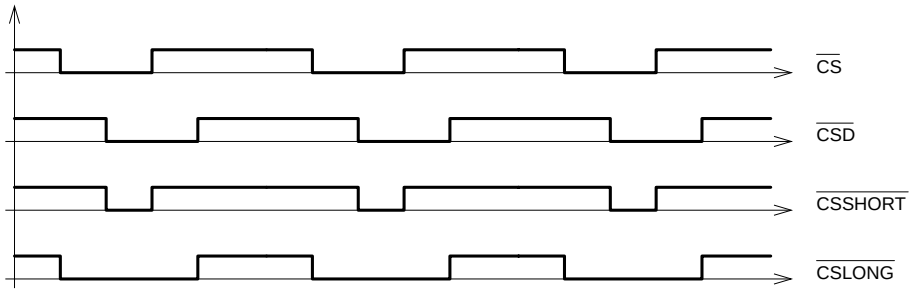


Abbildung A.2: Aus dem rekonstruierten *chip-select*-Signal $\overline{\texttt{CS}}$ und einer verzögerten Version $\overline{\texttt{CSD}}$ wird per Konjunktion der Takt $\overline{\texttt{CSLONG}}$ und per Disjunktion der Takt $\overline{\texttt{CSSHORT}}$ für die asynchrone Ablaufsteuerung gewonnen.

Die Steuerlogik kann neben dem Triggereingang auch einen Buszugriffszähler benutzen,

um den Umschaltzeitpunkt festzulegen. Um die ohnehin vorhandenen Bausteine optimal auszunutzen, werden die drei 16-bit Zähler im 8253-Baustein auf der Schnittstellenkarte im PC zu einem 32-bit Zähler kombiniert. Zähler A teilt die Takte durch 2^{16}, Zähler B zählt diesen geteilten Takt und Zähler C fängt erst an zu zählen, wenn Zähler B abgelaufen ist. Zähler B zählt damit die oberen und Zähler C die unteren 16 Bits. Wenn Zähler C abgelaufen ist, dann wird die SRAM/FIFO-Umschaltung aktiviert. Das Auslösen der Umschaltung durch eine Folge von Adressen kann mit entsprechender Zusatzlogik über den TRIG-Eingang erfolgen.

Die Programmierung des Lattice CPLD erfolgte mit Hilfe der LOG/iC2 Software der Firma ISDATA. Die folgende Quelldatei im LOG/iC2-Format beschreibt mit Logikgleichungen vollständig das Verhalten des Steuerchips:

```
*IDENTIFICATION
DS5002FP Crypt-Analyzer Logic
Markus Kuhn -- 1996-04-30
Informatik 3, Uni-Erlangen

-------------------------------------------------------------------------------

PC Control Signals:

PC0, PC1: 00 = triggered FIFO mode with analyzed SRAM (no writes after trigger)
          01 = triggered FIFO mode with DSCA SRAM (no writes after trigger)
          10 = CPU uses analyzed SRAM, no FIFO, recording mode
          11 = CPU uses DSCA SRAM, no FIFO intervention
PC2:      0  = send a clock pulse to counter A
PC3:      0  = send a clock pulse to counter B
PC4:      0  = send a clock pulse to counter C
PC5:      0  = DSCA SRAM chip select by PC (only with PC7 = 0)
PC6:      0  = DSCA SRAM write enable by PC (only with PC7 = 0)
PC7:      0  = enforce CPU reset, DSCA bus is free for PC access
          1  = allow CPU to run
PC8:      0  = write value into instruction FIFO

-------------------------------------------------------------------------------

*INTERFACE
IN:     CE_[1..4], RW, ADEN, ALE, CLK, MSEL;
IN:     TRIG, ARM_, EF_, DOFIFO, PC[0..8];
IN:     AOUT, BOUT, COUT;
OUT:    PRST, IRST;
OUT:    RWS, CES_[1..4];
OUT:    WRAM_, CSRAM_, GATE_, WINSF_, RINSF_, WFIFO_;
OUT:    FIFO, ARMED, COMP, EFR;
OUT:    ACK, AGAT, BCK, BGAT, CCK, CGAT;
OUT:    A[15..16], AA[15..16];
OUT:    CED_[1..4];
OUT:    DUMMY;

*LOCAL
CS_, CSD_, CSSHORT_, CSLONG_, CESHORT_[1..4], CELONG_[1..4];

*BOOLEAN-EQUATIONS
```

```
!!!!!!!!!!!!!!!!!!!!
!! Local variables !!
!!!!!!!!!!!!!!!!!!!!

! Chip select signal reconstructed from CPU outputs

CESHORT_[1..4] = CE_[1..4] + CED_[1..4];
CELONG_[1..4]  = CE_[1..4] & CED_[1..4];

CS_            = (MSEL & CE_1 & CE_2 & CE_3 & CE_4) + (/MSEL & CE_1);
CSD_           = (MSEL & CED_1 & CED_2 & CED_3 & CED_4) + (/MSEL & CED_1);
CSSHORT_       = (MSEL & CESHORT_1 & CESHORT_2 & CESHORT_3 & CESHORT_4) +
                 (/MSEL & CESHORT_1);
CSLONG_        = (MSEL & CELONG_1  & CELONG_2  & CELONG_3  & CELONG_4 ) +
                 (/MSEL & CELONG_1);

!!!!!!!!!!!!!!!!!!!!
!! Delayed signals !!
!!!!!!!!!!!!!!!!!!!!

CED_[1..4] = CE_[1..4];

!!!!!!!!!!!!!!!!
!! Registers !!
!!!!!!!!!!!!!!!!

! FIFO = 0  CPU can r/w access SRAM
! FIFO = 1  CPU gets instructions from and is recorded by FIFOs

FIFO         := (FIFO + COUT + (ARMED & (TRIG # COMP))) & /PC0;
FIFO.CLK     = CSD_;
FIFO.RS      = /PC7;

! COMP stores TRIG at falling edge of ARM_.

COMP         := TRIG;
COMP.CLK     = /ARM_;
COMP.RS      = /PC7;

! ARMED is 1 when COMP has been loaded by falling edge on ARM.

ARMED        := 1;
ARMED.CLK    = /ARM_;
ARMED.RS     = /PC7;

! Inverted EF_ value at previous rising edge of CS_;

EFR           := /EF_;
EFR.CLK       = CSD_;
EFR.RS        = /PC7;

!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Tristate output signals !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Reconstruct A15 and A16

A15          = (MSEL & CELONG_1 & CELONG_3) + (/MSEL & CELONG_3);
```

```
A16          = (MSEL & CELONG_1 & CELONG_2) + (/MSEL & CELONG_2);

AA15         = A15;
AA16         = A16;

AA[15..16].OE = ADEN & /CS_;

!!!!!!!!!!!!!!!!!!!
!! Output signals !!
!!!!!!!!!!!!!!!!!!!

! Control CPU reset.

PRST         = /PC7 + (EFR & /PC0);
IRST         = /(/PC7 + (EFR & /PC0));

! Control access to SRAM in analyzed system

RWS          = RW        + PC1 + /PC7 + FIFO;
CES_[1..4]   = CESHORT_[1..4] + PC1 + /PC7 + ((DOFIFO + /RW) & FIFO);

! Control access to DSCA SRAM

WRAM_        = (RW  + /PC1 + /PC7 + FIFO) & (PC6 + PC7) &
               (CSLONG_ + /PC0 + PC1 + /PC7);
CSRAM_       = (CSSHORT_ + /PC1 + /PC7 + ((DOFIFO + /RW) & FIFO)) &
               (PC5 + PC7) & (CSSHORT_ + /PC0 + PC1 + /PC7);

! Control FIFOs

WFIFO_       = CSSHORT_ + /FIFO + /PC7;
RINSF_       = CSLONG_ + /FIFO + /PC7;
WINSF_       = PC8;
GATE_        = CSD_ + /FIFO + /PC7 + /RW + /DOFIFO;

! Control 8253 counters (A: mode 2, B+C: mode 0)

ACK          = CS_  & PC2;
AGAT         = 1;
BCK          = AOUT & PC3;
BGAT         = 1;
CCK          = CS_  & PC4;
CGAT         = BOUT;

! Unused inputs

DUMMY        = CLK & ALE;

*END
```

Die Steuersoftware `dsca` wurde in Borland C++ 3.1 unter MS-DOS entwickelt und umfasst etwa 2000 Programmzeilen. Sie kann sämtliche in Kapitel 2 vorgestellten Testsequenzen selbstständig durchführen und erlaubt die Ausleseschaltung in allen Betriebsarten zu benutzen. Sie enthält verschiedene Testfunktionen für die Prüfung des Versuchsaufbaus sowie zur Durchführung von Versuchen, die Aufschluss über die Lage der Scheinzugriffe geben.

Die damit durchgeführten Vorversuche führten zu den in Kapitel 2 vorgestellten Testsequenzen. Die Kommandos der Steuersoftware werden beim Aufruf mit `dsca h` erklärt.

Die Ausleseschaltung wurde auf einer doppelseitig beschichteten Platine im Doppel-Europformat realisiert (Abbildung A.3). Auf den beiden Seiten der Platine befinden sich eine Masse- und eine 5-V-Versorgungsebene, um eine saubere Stromversorgung sicherzustellen. Alle Signalverbindungen wurden in Wire-Warp-Technik hergestellt (Abbildung A.4). Die Stromversorgung kann wahlweise über den Steuerrechner oder ein Netzteil erfolgen.

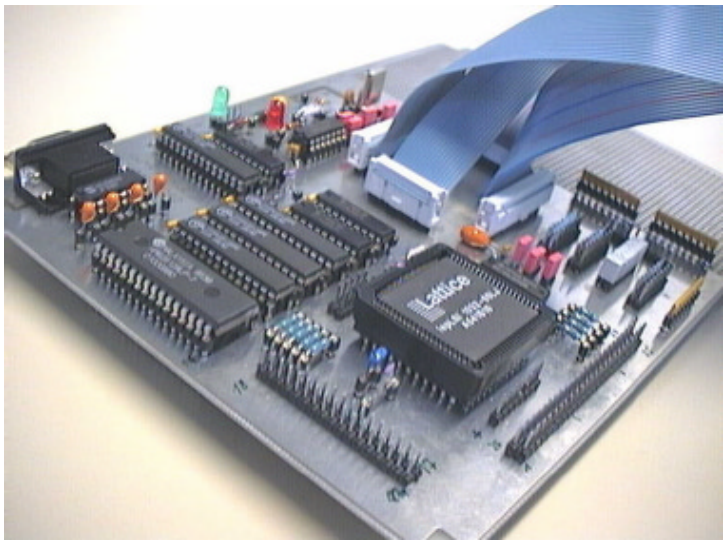


Abbildung A.3: Aufbau der Ausleseschaltung. Vorne sind die Kontakte zum Steuer-PC und der Lattice-Logikbaustein zu erkennen, links davon das SRAM, die FIFOs und die Bustreiber sowie die serielle Schnittstelle. Von der Mitte führen Flachbandkabel zum Testclip und im Hintergrund steht ein Rasterfeld für weitere Aufbauten wie etwa spezielle Triggermechanismen zur Verfügung.

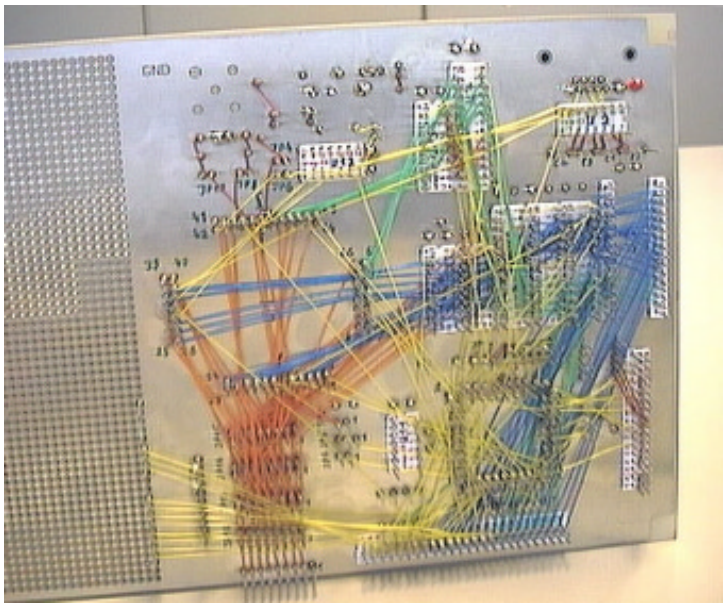


Abbildung A.4: Wire-Wrap Verdrahtung auf der Unterseite des Auslesesystems.

Da das untersuchte System bereits unter Batteriespannung steht und Daten enthält, müssen wir beim Anschluss der Ausleseschaltung sorgfältig auf die Reihenfolge der Vorgehensweise achten. Bei einem Ausleseversuch gehen wir im Normalfall in den folgenden Schritten vor:

S1 Anfertigung eines Schaltplans des untersuchten Systems. Insbesondere interessieren wir uns dafür, welche Schnittstelle des Prozessors für den Ausleseversuch benutzt werden kann, welche Peripherieanschlüsse gegebenenfalls entfernt werden müssen und wie die Resetlogik des Systems aufgebaut ist.

S2 Reinigung der Pins des Prozessors mit oxidlösendem Kontaktspray, um Kontaktschwierigkeiten mit dem Testclip zu reduzieren. Bei manchen Systemen muss erst ein Schutzlack von den Pins entfernt werden.

S3 Einbau eines Resetschalters zwischen $\overline{\texttt{VRST}}$ und `GND` in das zu untersuchende System, damit wir auch ohne aufgesetzten Testclip den Prozessor anhalten können. Dieser Schalter wird geschlossen.

S4 Die Steckbrücken der Ausleseschaltung werden für den Betrieb mit dem Testclip konfiguriert und der ausgewählte Prozessorport wird eingestellt.

S5 Wir verbinden die Masse des untersuchten Systems mit der Masse der Ausleseschaltung und schalten beide ein.

S6 Die Steuersoftware wird geladen und die Ausleseschaltung initialisiert. Der Steuersoftware wird der ausgewählte Port mitgeteilt (Kommando `p`) und sie wird für den Betrieb mit dem SRAM des untersuchten Systems konfiguriert (Kommando `m`).

S7 Um die $\overline{\texttt{CS}}$- und $\overline{\texttt{WE}}$-Pins der SRAMs vom Prozessor zu trennen, können wir wie folgt vorgehen: Ein etwa 50 cm langes Stück Wire-Wrap-Draht wird etwa 10 cm vom einen Ende auf einer Länge von etwa 8 mm abisoliert. An das andere Ende wird mit einer Crimp-Zange ein Steckkontakt für Pfostenstecker befestigt. Wie ziehen die 10 cm bis zur abisolierten Stelle hinter dem zu kontaktierten SMD-Pin durch, bis die abisolierten 8 mm genau hinter dem Pin herumlaufen. Nun verdrehen wir die beiden Enden des Drahtes dicht am Pin gegenseitig und befestigen diese Drahtschleife am Pin mit einem Tropfen Lötzinn. Anschließend verbinden wir das andere Ende mit dem Steckverbinder mit dem entsprechenden Kontakt von JP13 auf der Ausleseschaltung. Dann ziehen wir an den beiden verdrillten vom Chip weglaufenden Drahtenden nach oben während wir den Pin erhitzen. Der Pin löst sich von der Platine, und lässt sich durch weiteres Ziehen an den Drahtenden nach oben biegen. Auf diese Weise können wir sehr elegant die Verbindung zwischen SRAM-Chip und CPU trennen, ohne dabei den Pin offen zu lassen oder bleibenden Schaden auf der Platine zu verursachen. Diese Prozedur wiederholen wir mit allen SRAM-Pins die anzuschließen sind.

S8 Nun stecken wir den Testclip auf den Prozessor. Dabei sollte keinesfalls ein Kurzschluss zwischen den benachbarten Pins `SDI` und `VBAT` entstehen. Wenn entweder `VBAT` mit Masse verbunden wird oder `SDI` mit einer Versorgungsspannung, dann gehen die Daten unwiederbringlich verloren. Das Aufsetzen des Testclips ist nicht einfach und erfordert etwas Übung. Nach mehreren Versuchen können leicht Eckpins des Prozessors verbogen sein und das Aufstecken wird mangels korrekter Zentrierung noch schwieriger. Wenn der Einsatz eines Testclips nicht möglich ist können auch Drahtverbindungen hergestellt werden.

S9 Der Reset-Schalter wird geöffnet.

S10 Wir beginnen mit dem Testlauf T5 (`dsca`-Kommando `z`) um einen geeigneten Umschaltzeitpunkt zu finden. Gegebenenfalls muss erst noch eine Triggerschaltung aufgebaut werden, wenn eine zählerbasierte Umschaltung scheitern sollte.

S11 Mit dem `dsca`-Kommando `t` wird nach erfolgreicher Auswahl eines Umschaltzeitpunktes die Tabellarisierung der Datenbusverschlüsselung durchgeführt.

S12 Nun kann mit den Kommandos `c` und `d` der Programm- und Datenadressraum ausgelesen und im Intel-Hex-Format abgespeichert werden. Mit dem Kommando `s` können der Inhalt der Spezialregister und insbesondere die Speicherkonfiguration ermittelt werden. Das Kommando `y` erlaubt, die Funktionen EA und ED für weitere Untersuchungen der Verschlüsselungsalgorithmen zu tabellarisieren.

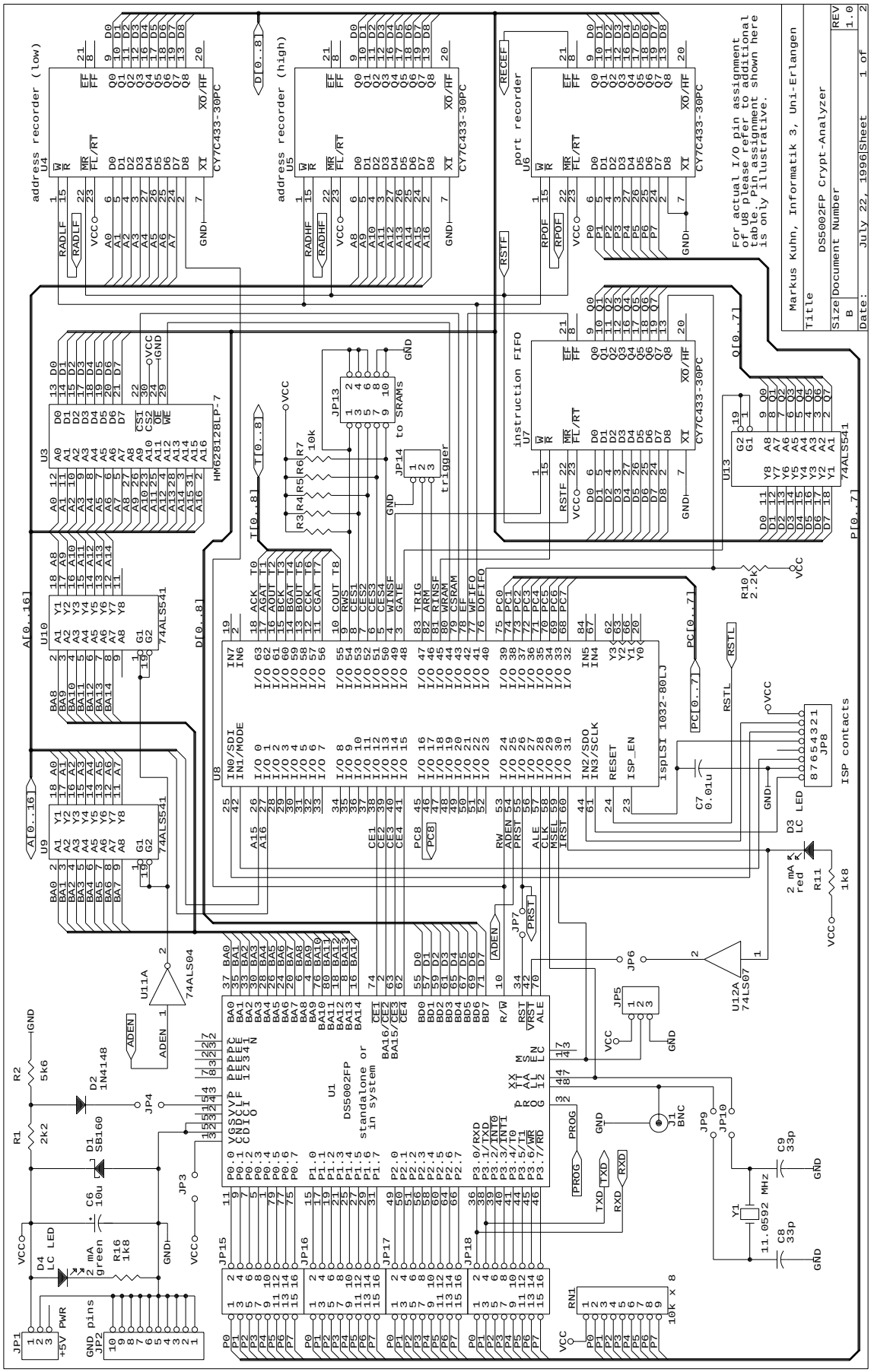
S13 Der Resetschalter wird wieder geschlossen um ein unkontrolliertes Weiterlaufen der CPU nach Abziehen des Testclips zu vermeiden.

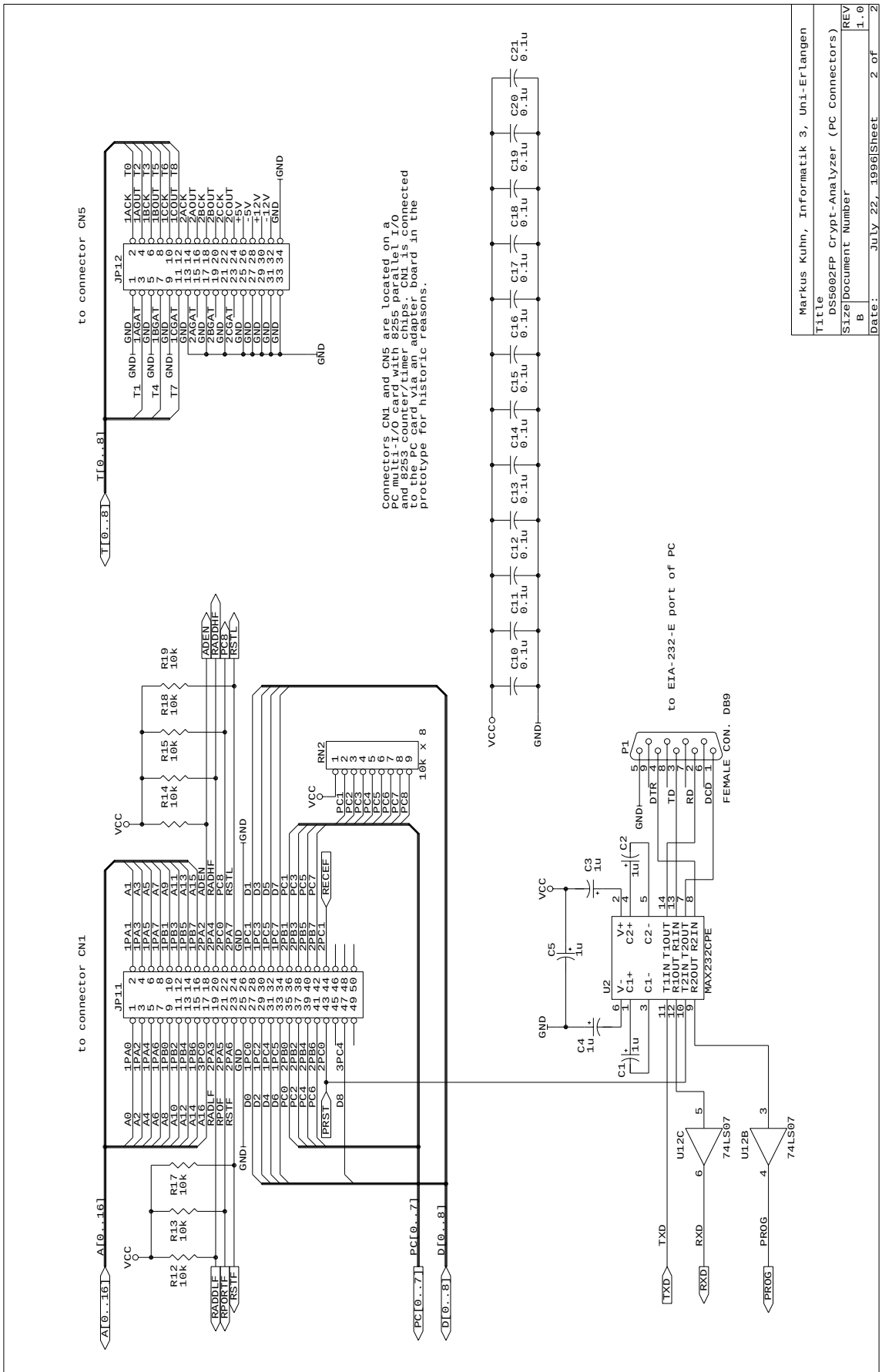
S14 Der Testclip wird abgezogen.

S15 Die angehobenen Pins werden erhitzt, die Drahtschleifen vorsichtig abgezogen und die SMD-Pins wieder auf die Platine gedrückt und angelötet. Der Lötkolben sollte dabei die ganze Zeit über einen *pull-up*-Widerstand mit +5 V verbunden sein, damit die $\overline{\texttt{CS}}$- und $\overline{\texttt{WE}}$-Pins des SRAMs ständig mit einem gültigen Signal versorgt werden und keine Schreibzugriffe durch das Ablöten ausgelöst werden.

S16 Wenn die SRAM-Pins wieder Kontakt mit der CPU haben, so kann die Versorgungsspannung abgeschaltet und die restlichen Verbindungen inklusive dem Reset-Schalter können entfernt werden.

Es empfiehlt sich, während dieser Prozedur den `VCC`-Stromverbrauch des DS5002FP im Auge zu behalten, um einen *latch-up* schnell erkennen zu können. Am Ende sollte auch die Spannung der Lithiumbatterie überprüft werden. Selbstverständlich müssen die üblichen ESD-Schutzmaßnahmen für den Umgang mit CMOS-Bauteilen beachtet werden. Wenn die hier geschilderte Vorgehensweise eingehalten wird, so sollte anschließend das ausgelesene System wieder voll betriebsbereit und die gespeicherten Daten völlig unverändert sein. Die einzigen zurückbleibenden Spuren sind zwei frisch verlötete Pins an jedem SRAM-Chip.


address recorder (low)
address recorder (high)
port recorder
instruction FIFO
CY7C433-30PC
HM628128LP-7
74ALS541
74ALS04
74ALS07
isplSI 1032-80LJ
DS5002FP
standalone or in system
Markus Kuhn, Informatik 3, Uni-Erlangen
DS5002FP Crypt-Analyzer
ISP contacts
For actual I/O pin assignment of U8 please refer to additional table. Pin assignment shown here is only illustrative.
July 22, 1996
GND pins
+5V PWR
LC LED
2 mA green
2 mA red
11.0592 MHz
10k x 8


to connector CN5
to connector CN1
JP12
JP11
T[0..8]
A[0..16]
D[0..8]
PC[0..7]
Connectors CN1 and CN5 are located on a
PC multi-I/O card with 8255 parallel I/O
and 8253 counter/timer chips. CN1 is connected
to the PC card via an adapter board in the
prototype for historic reasons.
VCC
GND
R12 10k
R13 10k
R14 10k
R15 10k
R17 10k
R18 10k
R19 10k
RN2
10k x 8
C1 1u
C2 1u
C3 1u
C4 1u
C5 1u
C10 0.1u
C11 0.1u
C12 0.1u
C13 0.1u
C14 0.1u
C15 0.1u
C16 0.1u
C17 0.1u
C18 0.1u
C19 0.1u
C20 0.1u
C21 0.1u
U2
MAX232CPE
U12C
U12B
74LS07
P1
FEMALE CON. DB9
to EIA-232-E port of PC
TXD
RXD
PROG
RADDLF
RPORTF
RSTF
ADEN
RADDHF
PC8
RSTL
RECEF
PRST
Markus Kuhn, Informatik 3, Uni-Erlangen
Title
DS5002FP Crypt-Analyzer (PC Connectors)
Size B
Document Number
REV 1.0
Date: July 22, 1996 Sheet 2 of 2

Literatur

[Abr91] Abraham, D. G. et al.: *Transaction Security System.* IBM Systems Journal, Vol. 30, No. 2, 1991, pp. 206–229.

[And96a] Anderson, Ross J.; Bezuidenhoudt, S. Johann: *On the Reliability of Electronic Payment Systems.* IEEE Transactions on Software Engineering, Vol. 22, No. 5, May 1996, pp. 294–301.

[And96b] Anderson, Ross; Kuhn, Markus: *Tamper Resistance – a Cautionary Note.* Preprint, submitted to USENIX Workshop on Electronic Commerce, Oakland, November 18–20, 1996.

[Bes79] Best, R. M.: *Microprocessor for Executing Enciphered programs.* U.S. Patent No. 4 168 396, September 18, 1979.

[Bes80] Best, R. M.: *Preventing Software Piracy with Crypto-Microprocessors.* Proc. IEEE Spring COMPCON 80, San Francisco, California, February 25–28, 1980, pp. 466–469.

[Bes81] Best, R. M.: *Crypto Microprocessor for Executing Enciphered Programs.* U.S. Patent No. 4 278 837, July 14, 1981.

[Bes84a] Best, R. M.: *Cryptographic Decoder for Computer Programs.* U.S. Patent No. 4 433 207, February 21, 1984.

[Bes84b] Best, R. M.: *Crypto Microprocessor that Executes Enciphered Programs.* U.S. Patent No. 4 465 901, August 14, 1984.

[Bih93] Biham, Eli; Shamir, Adi: *Differential Cryptanalysis of the Data Encryption Standard.* Springer, New York, 1993.

[Bly93] Blythe, S. et al.: *Layout Reconstruction of Complex Silicon Chips.* IEEE Journal of Solid-State Circuits, Vol. 28, No. 2, February 1993, pp. 138–145.

[CEN92] *Access control system for the MAC/packet family: EUROCRYPT.* Europäische Norm EN 50094, Europäisches Komitee für Elektrotechnische Normung (CENELEC), Dezember 1992.

[Cha83] Chaum, David: *Design Concepts for Tamper Responding Systems.* Advances in Cryptology, Proc. Crypto 83, Plenum Press 1984, pp. 387–392.

[Col87] Cole, Bernard C.: *Designer's Dream Machine – Dallas Semiconductor's Microcontroller updates itself on the fly.* Electronics, Vol. 60, No. 5, March 5, 1987, McGraw-Hill, pp. 53–57.

[Dal93] *Soft Microcontroller Data Book.* Dallas Semiconductor, Dallas, Texas, 1993.

[FIP77] *Data Encryption Standard.* FIPS PUB 46, Federal Information Processing Standards Publication, National Bureau of Standards, U.S. Department of Commerce, January 15, 1977.

[FIP94] *Security Requirements for Cryptographic Modules.* FIPS PUB 140-1, Federal Information Processing Standards Publication, National Institute of Standards and Technology, U.S. Department of Commerce, January 11, 1994.

[GEI94] *Der DALLAS DS5002 als Sicherheitsprozessor.* Untersuchungsbericht, debis Systemhaus GEI, Bonn, 1994-02-14.

[Gol86] Goldreich, Oded: *Towards a Theory of Software Protection.* Advances in Cryptology, Proc. Crypto 86, LNCS 263, Springer-Verlag, Berlin 1987, pp. 426–439.

[Int93] *MCS 51 Family of Microcontrollers Architectural Overview.* Intel, September 1993, Order Nr. 270251-004.

[ISO87] *Identification cards – Integrated circuit(s) cards with contacts.* ISO 7816, International Organization for Standardization, Geneva, 1987.

[Kah67] Kahn, D.: *The Codebreakers – The Story of Secret Writing.* Macmillan, New York, 1967.

[Ken81] Kent, Steven: *Protecting externally supplied software in small computers.* Ph.D. Dissertation, MIT Laboratory for Computer Science, MIT/LCS/TR-255, Cambridge, Massachusetts, March 1981.

[New90] *A system for controlling access to broadcast transmissions.* News Data Security Products, European Patent Application 0 428 252 A2, Europäisches Patentamt, 1990-09-17.

[Ost89] Ostrovsky, Rafail: *An Efficient Software Protection Scheme.* Advances in Cryptology, Proc. Crypto 89, LNCS 435, Springer-Verlag, Berlin 1990, pp. 610–611.

[Pri86] Price, W.L.: *Physical Security of Transaction Devices.* NPL Technical MEMO DITC 4/86, National Physical Laboratory, January 1986.

[Sch96] Schneier, Bruce: *Applied Cryptography – Protocols, Algorithms, and Source Code in C.* Second edition, John Wiley & Sons, New York, 1996.

[Wei87] Weingart, Steve H.: *Physical Security for the μABYSS System.* Proc. 1987 IEEE Symposium on Security and Privacy, April 27–29, 1987, Oakland, California, IEEE Computer Society Press, pp. 52–58.

[Whi87] White, Steve R.; Comerford, Liam: *ABYSS: A Trusted Architecture for Software Protection.* Proc. 1987 IEEE Symposium on Security and Privacy, April 27–29, 1987, Oakland, California, IEEE Computer Society Press, pp. 38–51.

[ZKA85] *Schnittstellenspezifikation für die ec-Karte mit Chip – Die elektronische Geldbörse.* Version 2.0, Zentraler Kreditausschuss, Bank-Verlag, Köln, 1995-03-10.