Ruhr-Universität Bochum
Lehrstuhl Kommunikationssicherheit
Seminar: IT-Sicherheit
Wintersemester 2003/2004

Hardwarebasierter Softwareschutz

Von: Nihat Gör
Matrikel-Nr.: 108 000 110 796
Betreut durch: Marcus Heitmann

Inhaltsverzeichnis

1 Allgemeine Betrachtungen

1.1 Hardwareschutz

In der heutigen Zeit ist es durch die Möglichkeiten die unter anderem das Internet bietet, sehr einfach raubkopierte Software illegal zu verbreiten. Um diesen Missbrauch zu unterbinden, wird neben Kopierschutzlösungen auch hardwareunterstützter Schutz der Software eingesetzt. Die Schutzwirkung wird dadurch erzielt, dass die Software nur mit der dazugehörigen Hardware, dem Dongle, funktioniert. Durch den Einsatz von Dongles ist zusätzlich zum Schutzaspekt möglich, die Software ohne Kopierschutz auszuliefern, um dem Benutzer die Freiheit zu geben Sicherungskopien anzulegen.

Dongles werden in verschiedenen Variationen hergestellt, welche in Ihrer Funktionsfähigkeit als auch in Ihrer Schnittstelle zum PC variieren. Die Funktionen die ein Dongle bietet, kann von der einfachen Meldung der Anwesenheit über Ver- bzw. Entschlüsselung von Programmsequenzen bis hin zur integrierten Funktion zur Generierung des Codeschlüssels reichen.

Die Komplexität der Funktion hat natürlich auch einen Einfluss auf die innere Schaltung des Dongles. Die entsprechend der Funktionsfähigkeit von der einfachen Schaltung mit Logikbausteinen bis hin zur komplexen Integrierten Schaltung in einem ASIC (**A**pplication **S**pecific **I**ntegrated **C**ircuit) variiert.

Die Sicherheit die mit Dongles erreicht werden kann ist ebenfalls von den oben erwähnten Funktionen abhängig. Der Schutz der durch die physikalische Anwesenheit des Dongles erzielt wird ist relativ gering, da die Abfrage des Dongles durch geübte Programmierer leicht softwareseitig umgangen werden kann. Die etwas schwieriger zu überwindende Variante des Dongles ist die, in dem die Daten verschlüsselt zum Dongle gesendet bzw. empfangen werden. Um diesen Schutz softwareseitig zu umgehen bedarf es einiger spezieller Kenntnisse über Kodierverfahren und besonderer Software, um im Quellcode nach den Dongleaufrufen zu suchen und diese dann dekodiert wieder in das Programm einzubringen, um somit auf das Dongle verzichten zu können.

1.1.1 Vorteile

- Die Software ist theoretisch ohne Dongle nicht zu betreiben.
- Der Anwender kann ohne Einschränkungen Sicherungskopien von der Software erstellen.
- Der Schutz gegen unbefugtes Benutzen und die damit verbundenen Vorteile, wie z.B. ungeschützte Betriebssysteme, können durch einfaches Entnehmen des Dongles erzielt werden.
- Es müssen keine Passwörter verwaltet werden, da der Zugriff nur durch das Dongle möglich ist.
- Die Software ist unabhängig vom Rechner zu überprüfen, vorausgesetzt das Dongle bietet diese Funktion.
- Eine gekaufte Softwarelizenz kann auf verschiedenen Rechnern genutzt werden, da ein paralleler Betrieb nicht möglich ist.

1.1.2 Nachteile

- Durch den Hardwareschutz steigen die Kosten für die Software.
- Das Dongle belegt eine Rechnerschnittstelle. Hierdurch können eventuell Kosten für eine zusätzliche Schnittstelle entstehen.
- Die Funktionen für den Donglezugriff die in Programmbibliotheken enthalten sind müssen auf das Betriebssystem angepasst sein. Dadurch können zeitliche Engpässe in der Verfügbarkeit der Software für neue Betriebssysteme entstehen.
- Komplexe Kodierverfahren können die Software verlangsamen, welches bei den jetzigen schnellen Prozessoren relativ unbedeutend ist.
- Der Nachbau der Hardware mit ASICs ist nahezu unmöglich. Dieser Punkt kann aus Sicht der Softwarehersteller auch als Vorteil gesehen werden.

1.2 Wirtschaftliche Aspekte des Softwareschutzes

In Relation zum erwarteten Umsatz mit der Software können höhere Kosten die durch den Einsatz von Dongles entstehen in einem vertretbaren Rahmen bleiben. Dies ist sowohl bei teurer als auch bei günstiger Software der Fall, da bei günstiger Software der Gewinn pro Produkt zwar deutlich geringer ist, aber der entgangene Gewinn durch illegale Kopien würde praktisch einen gleich großen oder sogar größeren Umsatzverlust mit sich bringen.

Bei dieser Überlegung ist natürlich anzumerken, dass der entgangene Gewinn durch illegale Kopien nicht direkt 1 zu 1 auf der positiven Seite zu Gunsten des Hardwareschutzes zu verbuchen ist. Denn die illegal kopierte Software ist für den Raubkopierer nahezu kostenlos und somit unterliegt das Programm auch nicht Kosten-Nutzen-Bewertungen. Daraus ist zu schließen, dass die illegal kopierten Produkte nicht in dem Maße auch gekauft würden, da sie dann anderen Bewertungskriterien unterliegen.

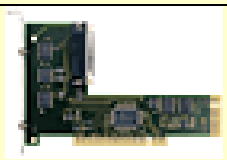
Zitat aus [10]: *„Nach Aussagen von Microsoft wird in Deutschland im Durchschnitt 40% der professionell genutzten Software raubkopiert.“*

Der Grund für die Raubkopien ist die Kostenersparnis, die sich bei kleineren Firmen durch den Nicht-Kauf der Software einstellt und bei Großunternehmen durch den geringeren Aufwand der Lizenzverwaltung begründet wird. Durch den Einsatz von intelligenten Kopierschutzvorrichtungen könnte rein theoretisch ein enormer Umsatzanstieg bei den Softwarehäusern erwirtschaftet werden.

2 Hardware

2.1 Ausführungsvarianten

Dongles werden in verschiedenen Bauformen, angepasst an die Schnittstelle an der sie eingesetzt werden und den Funktionen die sie bieten, ausgeführt. Die folgende Tabelle 2.1 gibt die verschiedenen Arten der Dongles mit ihren jeweiligen Eigenschaften wieder.

Bezeichnung	*Eigenschaft*	*Bild*
Memory-Dongle	– Anschließbar an die parallele Schnittstelle. – Integrierter Speicher. – Ermöglicht die Verwaltung und Freischaltung von Programmoptionen.	
PCI-Dongle	– Zur Nutzung als Dongle im Rechner.	
PCMCIA-Dongle	– Schutz von Software am Notebook. – Keine physikalische Verbindung mit anderen Peripheriegeräten und daher auch keine Kompatibilitätsprobleme dieser Geräte untereinander.	
Server-Dongle	– Zur Benutzung an der Parallel- oder USB-Schnittstelle für die Lizenzverwaltung im Netzwerk.	 
Standard-Dongle	– Zur Benutzung an der parallelen Schnittstelle.	
Twin-Dongle	– Sowohl an der seriellen wie auch an der parallelen Schnittstelle einsetzbar. – Aneinanderreihbarkeit von mehreren Dongles.	

Bezeichnung	*Eigenschaft*	*Bild*
USB-Dongle	– Plug & Play – Fähigkeit. – Einfache Benutzung am USB. – Es können bis zu 127 Stecker ohne Adresskonflikte am USB angeschlossen werden. – Keine physikalische Verbindung mit anderen Peripheriegeräten und daher auch keine Kompatibilitätsprobleme dieser Geräte untereinander.	

Tabelle 2.1: Donglevarianten

2.2 Beispiel für den internen Aufbau

Durch die Vielzahl der Funktionen die ein Dongle bieten kann, auf die in den nächsten Abschnitten näher eingegangen wird, kommen auch verschiedene Bauelemente zum Einsatz mit denen diese Funktionen realisiert werden können. Einfach ausgedrückt kann man sagen, je komplexer die Funktion, desto komplexer ist auch die eingesetzte Hardware.

An dieser Stelle soll exemplarisch der Standard-Dongle, wie unter [12] beschrieben, nur dargestellt werden. Dieser Dongle wird an der parallelen Schnittstelle des PCs eingesetzt. Er bietet die einfachen Möglichkeiten, wie Aufrufen, Lesen und Schreiben in den Dongle. Um diese Funktionen ausführen zu können, wird ein 4-Bit synchron Zähler vom Typ 74 HCT 161 eingesetzt. Damit der parallele Port weiter wie gewohnt arbeiten kann sind die Leitungen vom Eingang zum Ausgang des Dongles durchgeschliffen. Eine detaillierte Beschreibung des Dongles mit Programmierbeispielen ist unter [12] zu finden.

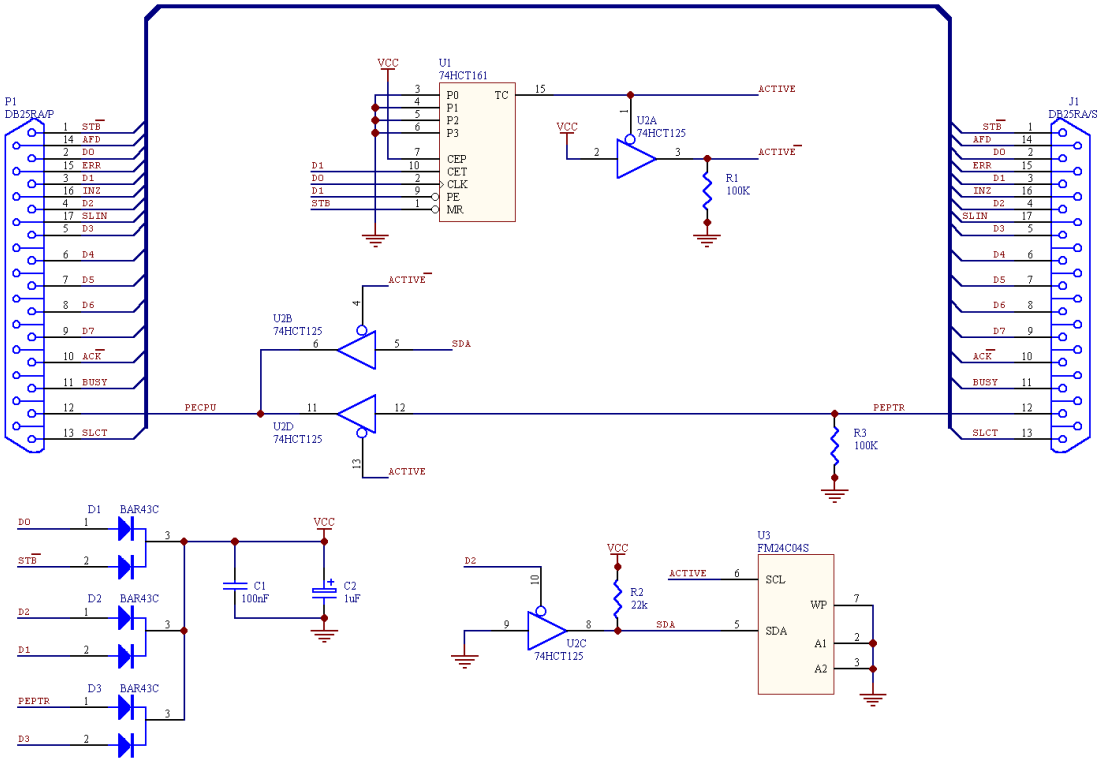


Abbildung 2.2: Schaltungsbeispiel eines Dongles vgl. [12]

3 Funktionsweise eines Dongle

3.1 Verschlüsselungsverfahren

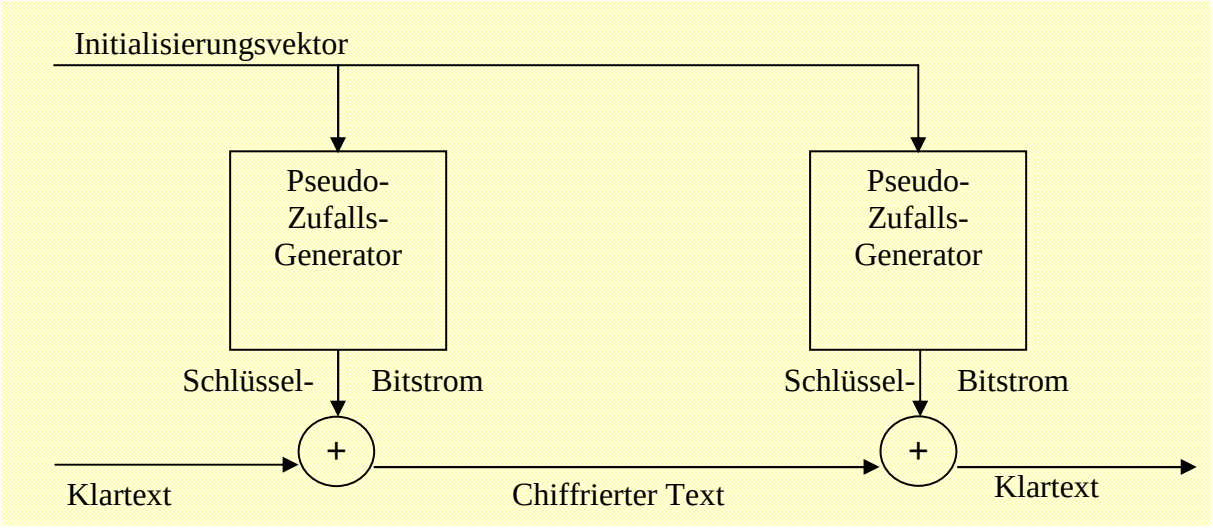
Bevor im Abschnitt 3.2 auf die Schutzmaßnahmen eingegangen wird, sollen in diesem Abschnitt die verschiedenen Verschlüsselungsverfahren betrachtet werden, die in Beziehung mit dem Dongle eingesetzt werden.

Bei der Verschlüsselung von Daten wird zwischen der Block-Chiffre oder der Stream-Chiffre unterschieden. Wie der Name schon sagt, werden bei Block-Chiffre die Daten zur Verschlüsselung in feste Blöcke zusammengefasst und bei der Stream-Chiffre werden die Daten im kontinuierlichen Strom verschlüsselt. Beide Verfahren sind bedingt durch ihre spezifischen Vor- und Nachteile für bestimmte Anwendungen unterschiedlich gut geeignet.

3.1.1 Stream-Chiffre

Hierbei wird ein Bitstrom von Daten wird über ein XOR (Addition modulo 2) mit einem Strom von Schlüsselbits verknüpft. Um aus den Chiffredaten wieder Klartext zu erhalten, muss der Vorgang nur mit den gleichen Schlüsselbits wiederholt werden.

Aus praktischen Gründen, wie z.B. die Übertragung des Schlüssels zum Empfänger und der Speicherplatzbedarf für den Schlüssel, kann der Schlüssel nicht dieselbe Länge wie der Datenstrom haben, so dass für den Strom der Schlüsselbits in der Regel ein parametrisierter Pseudo-Zufallsgenerator eingesetzt wird.



Die Parameter des Pseudo-Zufallsgenerators bilden dann den Schlüssel, da man über sie den zum Entschlüsseln benötigten Bitstrom erzeugen kann. Damit das Verfahren nicht vorhersagbar ist, darf sich der Strom von Pseudo-Zufallszahlen nicht wiederholen und auch nicht vorhersagbar sein.

Da bei kurzen Datenblöcken der Schlüssel immer gleich beginnt, besteht die Gefahr, dass potentielle Angreifer aus den vorhandenen Klartext- und Chiffre-Paaren den immer gleichen Schlüsselbitstrom rekonstruieren können. Deshalb wird noch zusätzlich ein Initialisierungsvektor eingeführt, der dem Pseudo-Zufallsgenerator einen Startwert vorgibt. Dieser Initialisierungsvektor muss nicht geheim sein und darf nicht mit dem Schlüssel verwechselt werden. Es ist jedoch wichtig, den Initialisierungsvektor regelmäßig zu verändern und diesen mit den Chiffredaten abzuspeichern, da er für die Dechiffrierung benötigt wird.

3.1.2 Block-Chiffre

Bei der Block-Chiffre wird ein Datenblock mit fester Größe über eine bijektive Abbildung in einen Chiffretext-Block der gleichen Größe abgebildet. Durch die Parameter der Abbildungsfunktion wird der Schlüssel der Block-Chiffre gebildet.

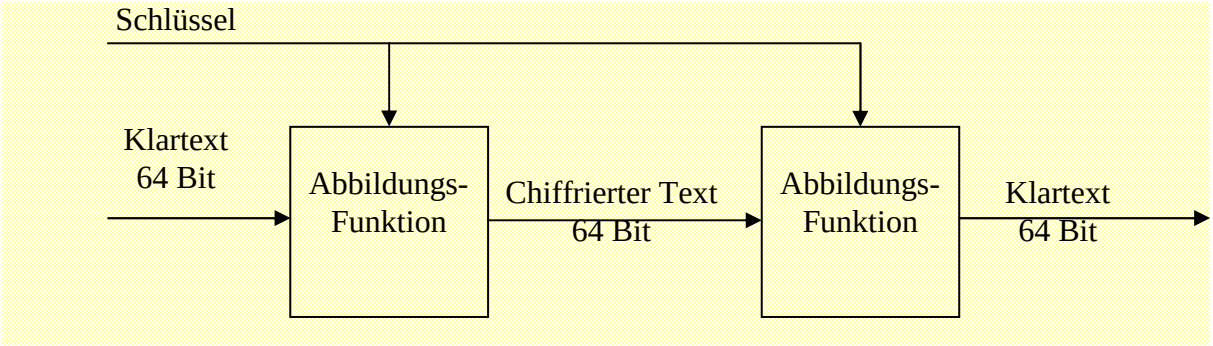


Abbildung 3.2: Block-Chiffre-Verfahren

Hieraus ergeben sich die folgenden Anforderungen an eine Block-Chiffre:

- Es ist darauf zu achten, dass die Blockgröße mindestens so groß ist, dass die Abbildung nicht durch eine Tabelle beschrieben werden kann. Es gilt allgemein als ausreichend, 64 Bit in einem Block zu verwenden, da es unrealistisch ist, eine Tabelle von 2^{64} Bit mit je 64 Bit Werten für jeden Schlüssel anzulegen.
- Der Versuch alle möglichen Schlüssel für ein Klartext-Chiffretext-Paar herauszufinden muss einen hohen Aufwand verlangen, welches durch einen großen Schlüsselraum zu erzielen ist.
- Die Abbildung muss so strukturiert sein, dass die Anzahl der zu analysierenden Schlüssel für ein Klartext-Chiffretext-Paar nicht über mathematische oder statistische Verfahren reduziert werden kann. Das heißt, dass es keine mathematische Abkürzung für die Schlüsselsuche geben darf. Die Mindestvoraussetzungen hierfür sind die folgenden Kriterien:

    - **Vollständigkeit:**

      Jedes Chiffretext-Bit muss von jedem Klartext-Bit abhängig sein. Diese Forderung wird durch das strikte Avalanche-Kriterium näher beschrieben: Bei Änderung eines Eingangsbits ist die Wahrscheinlichkeit für eine Änderung jedes Ausgangsbits etwa 50%.

    - **Nichtlinearität:**

      Zwischen den verschlüsselten und unverschlüsselten Daten darf es keinen linearen Zusammenhang geben.

In der Regel sind Block-Chiffren aufwendiger aufzubauen als Stream-Chiffren. Die beiden bekanntesten Verschlüsselungsverfahren DES (**D**ata **E**ncryption **S**tandard) und das Public Cryptosystem RSA (nach den Erfindern **R**ivest, **S**hamir, **A**dleman) sind beide Block-Chiffren.

Generell hat die Block-Chiffre gegenüber der Stream-Chiffre den Nachteil, dass wenn die Länge des Datensatzes das verschlüsselt werden soll nicht ein N-faches der Blocklänge ist, besondere Vorkehrungen getroffen werden müssen, um die Datenmenge in Grenzen zu halten.

Dieser Nachteil wird jedoch durch den Einsatz des Initialisierungsvektors der bei der Stream-Chiffre eingesetzt wird relativiert, da durch diese Maßnahme die Datenmenge auch erhöht wird.

3.1.3 Dongle-Chiffre

Das Entwicklungsziel der Dongle-Hersteller ist es, für Dongles ein Verschlüsselungsverfahren mit folgenden Eigenschaften zu finden:

I. Die Verschlüsselung muss leicht anwendbar sein.

II. Der Schlüssel muss durch die Hardware erzeugt werden.

III. Unterschiedliche Anfragen an die Hardware für unterschiedliche Daten.

Die III. Eigenschaft ist erforderlich, um den Schutzgrad proportional zu der Anzahl der Anfragen zu erhöhen. Dadurch wird erreicht, dass die Analyse des Dongles schwieriger ist als bei immer gleichen Antworten. Durch den Einsatz einer speziell konstruierten Feitel-Chiffre wird versucht, die Eigenschaft der unterschiedlichen Anfragen an die Hardware für unterschiedliche Daten zu erreichen. Diese auf einer Block-Chiffre basierende Chiffre verwendet reversible, zopfartige Verflechtungen der Daten mit einer bestimmten Anzahl von Unterschlüsseln, die die Entschlüsselung in umgekehrter Reihe durchführt.

Durch die intelligente Wahl der Abhängigkeit der Unterschlüssel von den Daten ergibt sich die Forderung, die Teilschlüssel über die Hardware zu generieren und bei der Entschlüsselung wieder aus den Chiffredaten zu rekonstruieren.

3.2 Maßnahmen zum Schutz

3.2.1 Angreifbarkeit

Jeder Schutz eines Programms kann grundsätzlich erkannt, analysiert und damit umgangen werden. Daher können Anbieter von Sicherheitsprodukten, die mit unüberwindbaren Maßnahmen werben, eigentlich nur als unseriös bezeichnet werden. Durch die mit hardwarebasierten Schutzprodukten zur Verfügung gestellten Maßnahmen ist es jedoch möglich, den Aufwand für die Analyse beliebig zu steigern.

Wenn eine Software wirksam geschützt werden soll, muss zuerst untersucht werden, wie ein potentieller Angreifer versuchen würde, die Schutzvorrichtung zu umgehen. Das Ziel bei der manuellen Einbindung von einem Dongle in die Software sollte sein, den Aufwand für das Umgehen des Schutzes so weit wie möglich zu erhöhen. Das Werkzeug des Angreifers ist in der Regel ein so genanntes Debug-Programm. Es ist davon auszugehen, dass der Angreifer die besten Werkzeuge zur Verfügung hat. Hierzu gehören vor allem Debugger, die den Prozessor im Protected Mode oder spezielle Debug-Hardware benutzen. Mit solchen Debuggern kann ein Programm im Einzelschrittverfahren abgearbeitet werden.

Der Angreifer tastet sich durch den Assemblercode, der vom Compiler erzeugt wurde. Dabei wird er versuchen, alle Anfragen an das Dongle zu finden. Nachdem der Angreifer die entscheidende Schutzmaßnahme erkannt hat, kann er das Programm so verändern, dass es ohne den Dongle funktioniert. Daraus wird offensichtlich, dass meist nicht eine besonders intelligente Maßnahme, sondern viele kleine triviale Maßnahmen den Aufwand für den Angreifer in die Höhe treiben.

Mit hardwarebasiertem Schutz kann eine Software auf einfache Weise so geschützt werden, dass ein Knackversuch ohne den dazugehörigem Dongle praktisch unmöglich ist, da für den Programmablauf notwendige Parameter, Daten oder Codeteile erst durch die Entschlüsselung durch den Dongle benutzbar werden. Wenn das Programm ohne Modul ablauffähig werden soll, müsste der Angreifer alle verschlüsselten Daten durch bereits entschlüsselte Daten ersetzen. Da der Angreifer die fehlende Information nicht erraten kann, muss er das Schutzmodul zur Verfügung haben.

Selbst wenn der Angreifer das passende Schutzmodul besitzt, und davon ist im Regelfall auszugehen, ist der Aufwand, alle richtigen Ergebnisse zu protokollieren und sie dann in das Programm einzufügen, extrem groß, wenn die Methode der Daten- und Parameter-verschlüsselung oft und auf das gesamte Programm verteilt angewendet wird. Dieser Aufwand kann über Scheinmanöver und Konsistenzchecks noch beliebig gesteigert werden.

3.2.2 Anwendung von High-Level Funktionen

Mit den Hochsprachenaufrufen kann der Schutz der Software individuell erhöht werden. Die folgende Aufzählung enthält eine Reihe von möglichen Maßnahmen, die angewendet werden sollten:

- Dongle-Modul initialisieren und terminieren.
- Vorhandensein des Dongle-Moduls vor der Entschlüsselung von großen Datenmengen überprüfen.
- Vorhandensein des Dongle-Moduls während des Programmablaufs überprüfen.
- Verschlüsseln von Programmteilen, internen und externen Daten.
- Sofortiges Verschlüsseln nach der Entschlüsselung und Verwendung (Zeitfensterverfahren).
- Physikalische Trennung und Auffächerung der Maßnahmen.
- Einbeziehen der Dongle-Memory-Option
- Maßnahmen nach Erkennen eines Angriffs ergreifen (verschleiern, verzögern etc.)
- Konsistenzchecks durchführen (Prüfsummen, Überprüfung des Kontrollmechanismus und der Ausführungszeit).

3.2.3 Verschlüsselung mit der Dongle-API

Die API (**A**pplication **P**rogramming **I**nterface) für Dongles ist die Schnittstelle zwischen einer zu schützenden Applikation und dem Dongle-Modul. Sie enthält in Form einer Dongle-Bibliothek sämtliche Funktionen zur Steuerung und Abfrage des Dongles.

Die API bietet den Vorteil, dass bei dem definierten Funktionssatz die Aufrufe unabhängig vom Betriebssystem sind. Ebenfalls sind die Funktionsnamen, außer wenn durch die Programmiersprache bedingt, unter allen Systemen identisch.

Die API ist in der Regel in zwei Ebenen unterteilt. Die Low-Level API und die High-Level API. In der Low-Level API sind alle Funktionen zur Ansteuerung des Dongles enthalten. Die für den Verbindungsaufbau zum Dongle notwendigen Treiber-Routinen sind ebenfalls darin enthalten und werden als Objekt-Dateien zur Verfügung gestellt. Die High-Level API ist als Zwischenschicht für eine einfachere und komfortablere Bedienung der API geschaffen worden.

3.2.4 Maßnahmen während des Programmablaufs

Die notwendigen Parameter für den Programmablauf sollten verschlüsselt sein und erst zur Laufzeit des Programms können dann die benötigten Daten oder Codeteile entschlüsselt werden. Durch die Möglichkeit, die benötigten Informationen in Laufzeit wieder zu entschlüsseln, kann dafür gesorgt werden, dass alle verschlüsselten Informationen nur für die kurze Zeit im Klartext vorliegen, in der sie wirklich benötigt werden (Zeitfenster-Verfahren). Die entschlüsselten Daten können z.B. verwendet werden, um Schleifen zu initialisieren, oder Parameter an eine Funktion zu übergeben. Das Ablegen entschlüsselter Werte als lokale Variablen auf dem Stack erschwert das Debuggen zusätzlich. Die Möglichkeit der Datenverschlüsselung sollte auch für externe Datendateien verwendet werden, wie z.B. Konfigurationsdateien oder Serialisierungsinformationen.

Vor jeder Entschlüsselung von großen Datenmengen sollte überprüft werden, ob das Modul mit der Kodierung noch vorhanden ist. Dadurch wird vermieden, dass eventuell wichtige Daten zerstört werden.

Die untere Auflistung der im Einzelnen beschriebenen Verfahren gibt die möglichen Maßnahmen während des Programmablaufs wieder:

- **Zeitfenster-Verfahren**

  Es besteht die Möglichkeit, benötigte Informationen erst zur Laufzeit zu entschlüsseln. Damit kann erreicht werden, dass bestimmte Informationen nur verschlüsselt im Programm enthalten sind und für kurze Zeit (in der diese tatsächlich benötigt werden) sowie nur teilweise - niemals aber gleichzeitig und vollständig - im Klartext vorliegen.

- **Auffächern der Maßnahmen**

  Die generelle Strategie eines baumartigen Auffächerns der Schutzmaßnahmen ist jeder noch so gut abgesicherten Einzelmaßnahme vorzuziehen. Die Abfragen für das Modul sollten auch in die tiefsten Menüebenen eingestreut sein.

- **Physikalische Trennung**

  Programmsequenzen zur Abfrage des Moduls sollten nicht immer gleich zusammenhängend programmiert werden, da sonst immer (fast) identischer Code erzeugt würde. Damit wären die Modulabfragen für den Angreifer leichter zu lokalisieren. Daher sollten diese Sequenzen im Sourcecode soweit möglich physikalisch aufgetrennt werden..

- **Zwang, das Programm kennen zu lernen**

  Meist kennt oder verwendet der Angreifer das Programm nicht. Daher kann es verhältnismäßig lange dauern, bis ein jugendlicher Angreifer z.B. ein komplexes Buchhaltungsprogramm bis in die kleinsten Detaildialoge kennen gelernt hat. Vor allem wird es immer schwierig für ihn sein, die korrekten Eingaben zu machen bzw. die Richtigkeit des Verhaltens eines solchen Programms zu überprüfen. Dazu kommt der psychologische Vorteil, dass der Angreifer nie genau weiß, wann er wirklich alle Fallen beseitigt hat.

- **Erzeugen von Rauschen**

  Hierzu werden an beliebigen Stellen im Programm viele Dongle Abfragen, bzw. Ver-/Entschlüsselungen durchgeführt. Als Argumente werden beliebige Werte übergeben, die zum Beispiel mit Zufallsgeneratoren, Zeitwerten, Zwischenergebnissen von Berechnungen usw. erzeugt werden. Natürlich wird keine dieser Abfragen ein plausibles Ergebnis zurückliefern. Dies ist aber sinnvoll, da ein Mitverfolgen aller Aufrufe praktisch unmöglich gemacht wird. Durch dieses Rauschen wird der Angreifer von den wirklich benötigten Daten abgelenkt.

- **Konsistenzchecks**

  Der Angreifer muss das Programm verändern, um es vom Sicherheitsmodul unabhängig zu machen. Dies bedeutet, dass er bestimmte Bytefolgen in dem Programm durch andere ersetzen muss. Eine der wichtigsten Maßnahmen, den Aufwand für die Umgehung eines Programmschutzes zu erhöhen, ist das Durchführen von Konsistenzchecks. Diese Maßnahme kann eigentlich nicht oft genug verwendet werden. Die Überprüfung sollte immer wieder auf verschiedene Arten durchgeführt werden, wie z.B.:

- Errechnen und Kontrollieren von unterschiedlichen Prüfsummen über empfindliche Programmteile. Auch die Prüfsummen sollten gegen Veränderung geschützt werden, z.B. durch Verteilung auf mehreren Variablen.
- Errechnen und Überprüfen bestimmter Teile der EXE-Datei, der Overlays, etc.
- Überprüfen von einzelnen entscheidenden Befehlen auf Veränderung. Für verschiedene Prüfsummen sollten auch verschiedene Algorithmen zur Berechnung verwendet werden. So könnten zum Beispiel im Speicher des Moduls (falls vorhanden) Konstanten abgelegt sein, die als Komponenten der Prüfsummen in die Berechnung mit einfließen.

- **Kontrolle der Kontrollfunktionen**

  Die Implementierung der Abfragen sollte vielschichtig und komplex sein. So sollten einerseits Routinen fragen: Ist das Modul noch vorhanden? Andere Routinen sollten feststellen: Sind die Abfrageroutinen noch vorhanden und funktionsfähig? Der kausale Zusammenhang dieser Konsistenzchecks lässt sich beliebig weit ausbauen. Dabei prüft eine gerade entschlüsselte Routine, ob eine andere, die gerade verschlüsselt sein müsste, auch wirklich verschlüsselt vorliegt. Durch eine Verschachtelung dieser Maßnahmen wird eine hohe Schutzwirkung erzielt.

- **Kontrolle von Ausführungszeiten**

  Die Ausführungszeit von Routinen mit bekannter Laufzeit sollten überprüft werden. Bei einer Ausführung im Single-Step- oder Trace-Modus eines Debuggers läuft ein Programm viel langsamer ab. Dies kann ausgenutzt werden, um festzustellen, ob ein Debug-Programm aktiv ist. Wenn ja, sollten möglichst subtile Maßnahmen getroffen werden, um den Angreifer nicht erkennen zu lassen, wann man seinen Angriff bemerkt hat.

3.2.5 Einbeziehen der Dongle Memory Option

Sollte der Dongle ein Speicher besitzen, so können die einzelnen Memory-Register ebenfalls in die Sicherheitsabfragen einbezogen werden. So könnten zum Beispiel Kundenname und Seriennummern im ROM-Bereich des Speichers abgelegt und während des Programmablaufs angezeigt werden. Auch eigentlich nicht benutzte Memory-Register können mit Zufallszahlen gefüllt werden, die die Applikation dann abfragt und nicht weiter verwendet. Es ist jedoch zu beachten, dass der RAM-Bereich prinzipiell von jeder Applikation beschrieben werden kann.

3.2.6 Maßnahmen nach Erkennen eines Angriffes

Wenn über eine der oben beschriebenen Methoden oder über Konsistenzchecks einen Angriff oder eine Veränderung des Programms erkannt wurde, gibt es eine Vielzahl von Möglichkeiten, das Programm unbrauchbar zu machen. Dabei sollten die getroffenen Maßnahmen so weit wie möglich kausal von der Erkennung des Angriffes "entkoppelt" werden. Dazu können die in den folgenden Abschnitten beschriebenen Maßnahmen verwendet werden.

- **Verzögerung:**

  Zeitliche Verschiebung der Maßnahme. Erst nach einiger Zeit (von Sekunden bis zu Tagen bei Ausnutzung des Systemdatums) geschieht etwas.

- **Verschleierung:**

  Verschleierung des kausalen Zusammenhangs. Eine kleine Änderung hat eine Veränderung von vielen anderen Werten zur Folge, von denen wiederum nur einer die eigentliche Maßnahme initiiert.

- **Verfälschungen:**

  Das Programm arbeitet weiter, aber falsch. Berechnungen weisen nicht korrekte Ergebnisse auf, die nicht sofort offensichtlich sind.

- **Einschränkungen:**

  Verschleierung durch schwer erkennbare Einschränkungen: Ein Angreifer merkt erst spät, dass ein Programm jetzt nur noch im eingeschränkten Demo-Modus läuft (z.B. nur wenige Datensätze, kein Ausdrucken, etc.).

4 Sicherheitsabschätzung

Um den Aufwand verschlüsselter Daten mit Dongles abschätzen zu können, müssen Annahmen über die Ressourcen des Angreifers gemacht werden. In dem folgenden Szenarium wird eine Aufwandsabschätzung vorgenommen.

Voraussetzungen:

1. Der Angreifer kann die Korrektheit des von Ihm erratenen Schlüssels eindeutig feststellen. Dies ist eine realistische Annahme, wenn es sich bei den unverschlüsselten Daten um lesbaren Text handelt.
2. Der Angreifer kennt jedoch nicht den vom Dongle-Hersteller geheim gehaltenen Algorithmus, der bei der Hardware-Schlüssel-Generierung durch den Dongle verwendet wird.
3. Der Angreifer verfügt über einen schnellen PC, mit dem eine Blockverschlüsselung mit Vergleich etwa alle 20 Mikrosekunden durchgeführt werden kann.

Aufwand:

Wenn der Algorithmus mit dem der Dongle arbeitet nicht bekannt ist, muss für jeden Datenblock erneut der ganze Schlüsselraum durchsucht werden, da der verwendete Sub-Schlüssel für jeden Datenblock verschieden ist.

In der hier angenommenen Routine werden für jede Verschlüsselung 48 Bit von der Hardware geholt. Deshalb müssen für jeden 64-Bit Block 2^{48} Möglichkeiten durchgerechnet werden. Wegen der Symmetrie der Chiffre und der Wahrscheinlichkeit, den richtigen Sub-Schlüssel nach 50% der Versuche zu finden, verringert sich diese Zahl um einen Faktor 4 auf 2^{46}. Der Aufwand für die Entschlüsselung eines 1-KB Datenblocks (128 x 64 Bit) wäre also:

$$\mathbf{128 \times 2^{46} \times 20\ \mu S = 1{,}8 \times 10^{11}\ s = \underline{5{,}7 \times 10^{3}\ Jahre}}$$

Entscheidend für die Sicherheit einer Chiffre ist nicht nur die Betrachtung der Größe des Schlüsselraums und der damit verbundene Aufwand für das Durchsuchen aller Schlüssel. Vielmehr besteht die Gefahr einer mathematischen Abkürzung, wenn eine Chiffre nicht nach kryptografischen Designkriterien entwickelt wurde, d.h. es gibt eine Methode, die den Aufwand für das Brechen der Chiffre weit unter den Aufwand für einen "exhaustive key search" sinken lässt. Bei dem Design der Dongle Routinen ist darauf zu achten, solche mathematischen Abkürzungen unmöglich zu machen. Dagegen sind die vielfach verwendeten Pseudo-Zufallszahlengeneratoren mit Schieberegistern durch Lösen von relativ einfachen linearen Gleichungssystemen zu brechen. Die Periodenlänge bzw. die Größe des Schlüsselraums dieser Chiffren ist dabei sehr groß.

5 Umgehung eines Dongle

Diesem Abschnitt liegt eine Anleitung aus dem Internet [11] zum Knacken von Dongles zu Grunde.

Um die Software zu analysieren werden Softwaretools wie Disassembler (z.B. IDA), Debug-Software (z.B. SoftICE), Hex-Programme (z.B. HIEW) benötigt. Der Angreifer braucht zur Umgehung des Schutzes der Software auch den zu der Software gehörenden Dongle.

Der erste Ansatzpunkt für den Angreifer ist die Identifikation des eingesetzten Dongles. Dies ist in den meisten Fällen nicht so schwierig, da die Daten des Dongles entweder auf dem Dongle selbst oder in einer der Dateien mit den Endungen *.dll oder *.vxd stehen.

Als nächstes werden vom Angreifer alle Daten, die über den Dongle verfügbar sind gesammelt. Diese Informationen sind auf der Homepage der Hersteller zu finden. In den meisten Fällen wird der volle Dongle-API mit Quellcode, Demosoftware und Beispielen zur Einbindung in die zu schützende Software zum Download bereitgestellt. Diese Informationen werden vom Angreifer intensiv studiert.

Der Weg, der zur Umgehung des Dongles vom Angreifer genutzt wird, ist das nicht versucht wird die Hardware zu modifizieren, sondern vielmehr die Dongleaufrufe zu identifizieren und die vom Dongle auf diese Aufrufe hin an das Programm gesendeten Daten in der Software selbst zu implementieren, um so das Dongle zu umgehen.

Die in der Anleitung aus dem Internet betrachteten Dongles von vier verschiedenen Programmen waren ohne viel Mühe und Zeitaufwand relativ einfach zu umgehen, denn es wurden die speziell von den Hardwareherstellern vorbereiteten Funktionen nicht oder nur teilweise genutzt.

Um die Möglichkeiten die ein Dongle bietet voll ausnutzen zu können, müssen die Soft- und Hardwarehersteller enger zusammenarbeiten und das Informationsangebot an dritte wie z.B. über das Internet muss begrenzt werden.

6 Literaturliste

[1] Aladdin Dokument: Addendum Tech-Doc 11/02 D
HL-mm (11/2002) 3.0-00353
Revision: 3.00
Datum: 28.11.2002

[2] Aladdin Dokument: Hardlock Begriffe
HL-mn (10/2000) 1.2 A1356
Revision: 6.2
Stand: 01.10.2000

[3] Aladdin Dokument: Hardlock High-Level API
HL-mn (10/2000) 2.1 A1355
Revision: 7.1
Stand: 01.10.2000

[4] Aladdin Dokument: Hardlock Bistro
HL-mn (10/2000) 1.2 A1352
Revision: 2.2
Stand: 01.10.2000

[5] Aladdin Dokument: Hardlock HL-Crypt
HL-mn (10/2000) 2.1 A1354
Revision: 7.1
Stand: 01.10.2000

[6] Aladdin Dokument: HL-Server
HL-mn (11/2002) 1.1-A03909
Revision: 7.2
Datum: November 28, 2002

[7] Aladdin Dokument: Hardlock Technik
HL-mn (10/2000) 1.2-A1351
Revision: 6.2
Stand: 01.10.2000

[8] Aladdin Dokument: Kurzanleitung

[9] http://global.bsa.org/germany/presseecke/2003/Bs043-02.html
PRESSEMELDUNG RESSORT SOFTWARE/ WIRTSCHAFT/ INTERNET
Nr. Bs043-02, 2. April 2003

[10] http://www.ad.or.at/text/67.htm
DIR 17/12 (Vol. 4, Jahrgang 1992, Heft 5, S. 146)
Schrankenlose Sicherheit
Softwareklau und Softwareschutz

[11] http://66.98.132.48/fravia/zee__4.htm
Zen and the Art of Dongle Cracking

[12] http://www.ping.be/~ping0751
I^2C Software Drivers