**Hochschule
für Technik, Wirtschaft
und Kultur Leipzig (FH)**

Fachbereich Informatik

Entwurf und Implementierung eines zentralisierten IP-basierten Ressourcen-Abfrage-Systems für Windows

Diplomarbeit von

Matthias Hönemann

Leipzig, Oktober 2002

Inhaltsverzeichnis

0. Einleitung

Da die Netzwerke immer größer werden und die Anzahl der in ihnen vorkommenden Rechner ständig wächst, wächst auch der Aufwand der Administration und Überwachung dieser Rechner. Eine Übersicht über Hardwarebestückung und Konfiguration der einzelnen Rechner zu erlangen, ist mit einem großen Aufwand verbunden. Eine komfortable Lösung dieses Problems bestünde darin, eine solche Übersicht zentral von einer Stelle aus über diese Rechner zu erhalten.
Zwar gibt es Softwaresysteme, die eine solche Funktionalität besitzen, wie z.B. Tivoli von IBM, jedoch sind diese Systeme zum einen sehr teuer (mehrere hunderttausend bis Millionen Euro, je nach Funktionsumfang) und überladen an Funktionen und müssen zum anderen so stark angepasst werden, dass eine eigene Implementierung dieser Funktionalität nicht mehr Aufwand und vor allem deutlich weniger Kosten bedeuten. Des Weiteren kann eine eigene Implementierung beliebig verändert werden, um z.B. die ressourcen-schonendste Lösung zu entwickeln.
Aus diesem Grund wurde ein Konzept entworfen und eine entsprechende Implementierung realisiert, das über eine Client-Server-Architektur Informationen über einen Rechner ausliest. Das System bedient sich dabei der Windows Registry, verschiedener Methoden, die in Windows-DLLs gekapselt sind sowie des BIOS-Roms. In diesem Rahmen wurde eine DLL entwickelt, in der Methoden zum Auslesen dieser Ressourcen-Informationen gekapselt wurden. Somit wird ein hoher Wiederverwendungswert erreicht, da diese DLL in beliebige Anwendungen integriert werden kann. Um die zu übertragenden Ressourcen-Informationen vor unberechtigtem Lesen und unbemerktem Ändern zu schützen, falls die Daten z.B. über das Internet übertragen werden, beinhaltet das System einen Datenschutz mit Hilfe verschiedener Verschlüsselungsmethoden und -techniken. Weiterhin können die Ressourcen-Informationen, die mit Hilfe des Systems ausgelesen werden, in einer XML-Datei gespeichert werden. Da XML ein einheitlicher Standard ist und als Universallösung der Softwareintegrationsprobleme gesehen wird und immer mehr Anwendung in Programmpaketen findet, können die so gespeicherten Ressourcen-Informationen problemlos weiterverarbeitet werden.

Die vorliegende Arbeit gibt im ersten Kapitel einen Überblick darüber, wo und wie Ressourcen-Informationen eines Rechners gespeichert werden und auf welche Art und Weise diese ausgelesen werden können. Dabei wird vor allem auf die Registry von Windows eingegangen, wobei die Speicherung der Angaben der Konfiguration, sowie der Hardware- und Softwarebestückung eines Rechners im Vordergrund stehen. In diesem Abschnitt erhält der Leser einen fundierten Überblick über die Struktur der Registry. Im zweiten Teil dieses Kapitels werden Methoden, sowie Windows-DLLs, in denen diese Methoden gekapselt sind, mit Hilfe derer Ressourcen-Informationen ausgelesen werden können, die nicht in der Registry gespeichert sind, erläutert. Im letzen Teil des ersten Kapitels wird auf das BIOS eines Rechners eingegangen. Hier wird ebenfalls vordergründig erläutert, auf welche Art und Weise Ressourcen-Informationen im BIOS-Rom gespeichert sind und wie diese ausgelesen werden können.

Das zweite Kapitel dieser Arbeit gibt einen Überblick über Verschlüsselungstechniken. Dabei wird auf Vor- und Nachteile der verschiedenen Techniken eingegangen und beschrieben, wie diese Techniken am besten miteinander kombiniert werden können, um ein optimales Maß an Sicherheit und Performance gewähren zu können. Des Weiteren wird ein Vertreter jeder Verschlüsselungstechnik, also eine Verschlüsselungsmethode, die am häufigsten in der Praxis eingesetzt werden und als die sichersten gelten, in Punkto Aufbau, Funktionsweise und Sicherheit näher erläutert. Diese Methoden werden im Ressourcen-Abfrage-System verwendet.

In Kapitel drei wird das Konzept des Ressourcen-Abfrage-Systems erläutert. Dabei wird unter anderem auf den Aufbau von Server-, Clientprogramm und der entwickelten DLL, das verwendete Kommunikationsprotokoll, die verwendete Verschlüsselung, den Aufbau der Konfigurationsdateien des Systems, den Ablauf zwischen Server- und Clientprogramm, sowie die Bedienung der Anwendungen eingegangen.

Kapitel vier beinhaltet die Erläuterung der Implementierung des Abfrage-Systems. Hierbei wird speziell darauf eingegangen, welche Programmier-techniken verwendet wurden und in welchem Format Ressourcen-Informationen, Konfigurationsdateien des Systems, sowie verschlüsselte Übertragungsdaten gespeichert sind. Des Weiteren werden der Aufbau und der

Umgang mit der entwickelten DLL näher erläutert, wobei gezeigt wird, wie einfach die DLL, ohne Kenntnis der inneren Struktur, transparent unter den Betriebssystemen Windows 95, 98, ME, 2000 und XP, verwendet werden kann. Anschließend werden die Units und Klassen von Client-, Serverprogramm und der entwickelten DLL sowie die verwendeten Komponenten aufgelistet.

In Kapitel fünf werden anhand von Screenshots Beispielabfragen des Ressourcen-Abfrage-Systems demonstriert. Dabei werden die dargestellten Informationen der Screenshots von Server- und Clientprogramm erläutert.

Kapitel sechs zeigt anhand von Lastmessungen, wie stark das Abfrage-System CPU, Netzwerk, und Arbeitsspeicher belasten. Dabei wird auch der Einfluss verschiedener Schlüssellängen, der verwendeten Schlüssel der Verschlüsselungsmethoden, auf die Belastung von CPU, Netzwerk und Arbeitsspeicher untersucht. Abschließend wird analysiert, welche Faktoren, welche Bereiche am stärksten belasten und worauf bei der Wahl der Schlüssellänge geachtet werden sollte.

Im letzen Kapitel wird eine Zusammenfassung der Arbeit gegeben. Des Weiteren werden Erweiterungsmöglichkeiten erläutert, um das Ressourcen-Abfrage-System zu verbessern bzw. zu optimieren.

1. Ressourcen eines Rechners

1.1. Die Windows Registry

Dieser Abschnitt gibt einen detaillierten Überblick über die Registry[1] von Windows. Dabei wird speziell auf die Speicherung von Hardware- und Softwareressourcen in der Registry eingegangen. Folgende Windowsversionen werden dabei untersucht:

- Windows 95
- Windows 98
- Windows ME (Millennium Edition)
- Windows 2000
- Windows XP (eXPerience)

Da die Registry von Windows ME auf Windows 98 aufbaut und die von Windows 98 auf Windows 95, wird die Registry dieser Windowsversionen zusammen betrachtet, bezeichnet als Windows 9x/ME. Ebenso verhält es sich mit der Registry von Windows 2000 und Windows XP, wobei die Zusammenfassung als Windows 2000/XP bezeichnet wird.

Da der Inhalt der Registry seitens Microsoft kaum dokumentiert ist, stützen sich Autoren, die über Teile des Inhalts der Registry schreiben, meist auf Vermutungen, die sich zum Teil als fehlerhaft erweisen. Weiterhin sind Teile der Registry gar nicht dokumentiert.
Die im Rahmen dieser Arbeit durch den Autor herausgefundenen Informationen über Teile des Inhalts der Registry, die nicht publiziert sind, sind die Abschnitte, die als Quellenangabe die Bezeichnung [*] tragen. Dabei wurde vorwiegend das Programm *Regmon* von der Firma *Sysinternals* [Sys02] eingesetzt, das die Zugriffe auf die Registry über die Registry-API überwacht.

[1] Registrierung (deutsch)

1.1.1. Was ist die Windows Registry?

Die Windows Registry ist die zentrale Datenbank von Windows. In ihr sind z.B. folgende Informationen gespeichert [Wal01][Hor02]:

- Erfassung der Dateien nach ihren Typen (Dateiendung)
- Zuordnung der Applikationen zu bestimmten Dateitypen
- Verwaltung und Speicherung der Rechner-Hardware einschließlich Plug and Play-Spezifikationen
- Verwaltung und Speicherung mehrerer Nutzer- und Systemkonfigurationen
- Konfiguration der installierten Applikationen
- COM-,OLE- und ActiveX-Informationen
- Systemeinschränkungen für bestimmte Anwender (Anwendergruppen)

1.1.2. Warum wurde die Registry eingeführt?

Seit der ersten Version von Windows werden hardware- und softwarespezifische Informationen und Einstellungen in INI-Dateien gespeichert. Diese Dateien sind im Textformat gespeichert, also mit einem einfachen Texteditor bearbeitbar. Eine INI-Datei ist in mehrere Abschnitte unterteilt, dessen Bezeichnungen in eckige Klammern eingeschlossen sind. Die eigentlichen Datenwerte werden in Schlüsselvariablen der Form *Schlüsselvariable=Wert* gespeichert. Die Konfigurationsdatei *Win.ini* enthält z.B. folgenden Abschnitt [Bor98]:

```
[Desktop]
Wallpaper=(None)
TileWallpaper=1
WallpaperStyle=0
```

Zur Speicherung von Systeminformationen und -konfigurationen wurden folgende Dateien verwendet:

- Autoexec.bat
- Config.sys
- Win.ini
- System.ini
- Control.ini
- Lanman.ini
- Protocol.ini

Dazu kamen noch die INI-Dateien der installierten Anwendungen.
Seit Windows 95 wird die Registry den INI-Dateien vorgezogen. Jedoch stellt sich die Frage, welche Beweggründe die Entwickler von Windows hatten, dies zu tun. Schließlich sind INI-Dateien so aufgebaut, dass sie mit einem einfachen Texteditor verändert werden können (die Registry ist in einem speziellen Format gespeichert, das nicht mit einem Texteditor bearbeitet werden kann, siehe Abschnitt 1.1.6). In den ersten Versionen von Windows funktionierte das System der INI-Dateien zufrieden stellend, allerdings gab es bei Windows 3.1 und 3.11 einige Probleme [Bor98][Hor02]:

- INI-Dateien werden nur bis zu einer Größe von 64 KB unterstützt.
- Damit *Win.ini* und *System.ini* nicht zu groß wurden, benutzen Entwickler oft eigene INI-Dateien, so dass das System unübersichtlich und schwer zu verwalten war.
- Alle in den INI-Dateien gespeicherten Abschnitte werden sequentiell gespeichert, womit sich der Zugriff auf diese Dateien verlangsamt.
- Des Weiteren gibt es keine Betriebssystemunterstützung für den Zugriff auf INI-Einträge über das Netzwerk. Hierfür sind keine APIs vorhanden.

Somit wurde bei Windows NT und Windows 95 ein neues System zur Speicherung von Systeminformationen, die Registry, eingeführt. Zur Gewährleistung der Abwärtskompatibilität wurde das System der INI-Dateien weiter unterstützt.

Die Windows Registry bietet die folgenden Vorteile gegenüber den INI-Dateien [Mic98][Hor02]:

- Die Größe ist nicht mehr auf 64 KB beschränkt.
- Die Registry ist nach einem Systemabsturz einfach restaurierbar, da alle Daten zentral in einer Datenbank gespeichert werden.
- Neben Zeichenketten können auch binäre Daten gespeichert werden.
- Ein Satz von netzwerkunabhängigen Funktionen zum Verändern und Auslesen von Registry-Informationen erlaubt es, Systemadministratoren, die Systemkonfiguration entfernter Rechner zu überprüfen.
- Durch benutzerspezifische Konfigurationen, welche auf einem zentralen Server gespeichert werden können, ist es dem Benutzer möglich, Zugriff auf seine Einstellungen von einem beliebigen Rechner aus zu erhalten, auf dem er sich anmeldet.
- Durch die zentrale Speicherung von Informationen ist jeder Anwendung garantiert, dass sie einheitliche Informationen des Systems erhält.
- Durch die hierarchische Struktur der Registry ist eine höhere Übersichtlichkeit als bei den INI-Dateien, die nur aus zwei Ebenen (Abschnitte und Schlüsselvariablen) bestehen, gewährleistet.

Die Windows Registry besitzt nicht nur Vorteile, sondern auch einige Nachteile [Hor02]:

- Eine Applikation kann durch fehlerhafte Einträge in der Registry das Funktionieren des Betriebssystems gefährden.
- INI-Dateien können mit einem beliebigen Texteditor, die Registry dagegen kann nur mit dem Registrierungseditor oder über spezielle API-Funktionen bearbeitet werden.
- Da der Inhalt der Registry seitens der Entwicklerfirma Microsoft kaum dokumentiert ist, stützen sich Autoren, die über Teile des Inhalts der Registry schreiben, meist auf Vermutungen, die sich zum Teil als fehlerhaft erweisen.

1.1.3. Der Aufbau der Registry

Die Registry ist hierarchisch aufgebaut. Die Wurzel dieser Struktur bilden sechs Hauptschlüssel, genannt Hive Keys (HKEYs) [Rob00][Wor99]:

1. HKEY_CLASSES_ROOT
2. HKEY_CURRENT_USER
3. HKEY_LOCAL_MACHINE
4. HKEY_USERS
5. HKEY_CURRENT_CONFIG

6.1 HKEY_DYN_DATA (nur Windows 9x/ME)

6.2 HKEY_PERFORMANCE_DATA (nur Windows 2000/XP)

Der Hauptschlüssel HKEY_DYN_DATA existiert nur bei Windows 9x und ME, der Hauptschlüssel HKEY_PERFORMANCE_DATA nur bei Windows 2000 und XP.

Jeder dieser Hauptschlüssel enthält Unterschlüssel (SubKeys) und diese ebenfalls weitere. Jedoch werden alle Unter-Unterschlüssel nur Unterschlüssel genannt. Die Bezeichnung sagt also nichts über die Hierarchieebene aus, in der sich der Unterschlüssel befindet. Jeder Unterschlüssel kann mehrere Einträge (Entries) und ihnen zugeordnete Werte (Values) enthalten. Dem Unterschlüssel selber kann auch ein Wert zugeordnet werden, welcher im Registry-Bearbeitungsprogramm *RegEdit* als *(Standard)* dargestellt ist. In Abbildung 1 ist die Hierarchie der Registry mit Hilfe des Registrierungseditors *RegEdit* zur Verdeutlichung der Grundbegriffe dargestellt [Wal01].

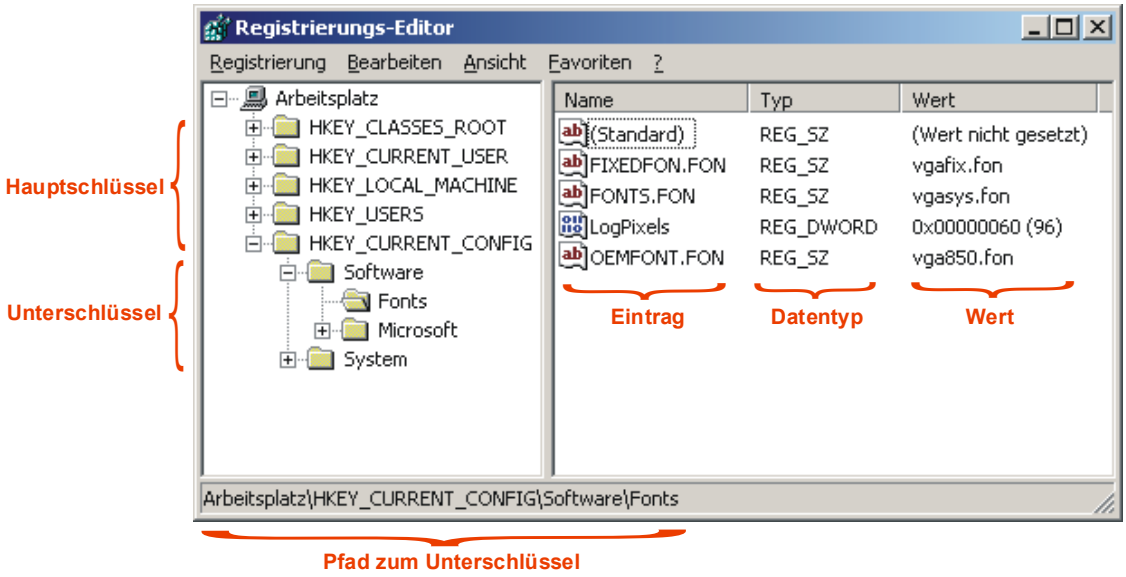


Abbildung 1 : graphische Darstellung der Registry mit Hilfe des Registrierungseditors zur Verdeutlichung der Grundbegriffe

Tabelle 1 enthält einen allgemeinen Überblick über die Inhalte der Hauptschlüssel. Eine detaillierte Beschreibung dieser Inhalte erfolgt in Abschnitt 1.1.8 [Wal01].

HKEY	Abkürzung	Inhalt
HKEY_LOCAL_MACHINE	HKLM	Konfigurationsdaten für Hardware, System, Treiber, Netzwerk, Dienste, Sicherheit, Software
HKEY_USERS	HKU	Benutzerprofile, benutzerbezogene Einstellungen der Arbeitsumgebung
HKEY_CURRENT_USER	HKCU	Benutzerprofil des aktuell am System angemeldeten Nutzers
HKEY_CURRENT_CONFIG	HKCC	Konfigurationsdaten der aktiven Hardwarekonfiguration
HKEY_CLASSES_ROOT	HKCR	Definition von Dateiendungen bzw. Dateitypen sowie den zugehörigen Applikationen (wichtig für OLE und ActiveX)
HKEY_DYN_DATA (nur Windows 9x/ME)	HKDD	wird dynamisch beim Systemstart erzeugt und existiert nur im Arbeitsspeicher des Rechners; enthält Konfigurations- und Leistungsdaten zum aktuellen Rechner
HKEY_PERFORMANCE_DATA (nur Windows 2000/XP)	HKPD	wie HKEY_DYN_DATA, enthält jedoch nur Leistungsdaten

Tabelle 1 : allgemeiner Überblick über die Inhalte der Hauptschlüssel (HKEYs) der Registry

Die Hauptschlüssel HKLM und HKU werden als Master-HKEYs bezeichnet. HKLM enthält alle Hardwarekonfigurationen, HKU alle Userkonfigurationen. Da

immer nur eine Hardwarekonfiguration und eine Userkonfiguration zur gleichen Zeit aktiv sein können, wird die aktive Hardwarekonfiguration mit HKCC und die aktive Userkonfiguration mit HKCU verlinkt. Somit muss sich eine Applikation nicht darum kümmern, welche Konfiguration aktiv ist. HKCR ist ein Link (Verweis) auf *HKLM\Software\Classes* und ermöglicht ein einfaches Zugreifen auf die Definition von Dateierweiterungen und ihre Zuordnungen zu Applikationen. Des Weiteren sichert dieser Schlüssel auch die Abwärtskompatibilität zu Windows 3.1-Applikationen. Abbildung 2 zeigt das Anlegen von Links beim Starten vom bzw. nach dem Anmelden am Windows 9x/ME- bzw. Windows 2000/XP-System. Die Hauptschlüssel HKDD bzw. HKPD werden aus dem Arbeitsspeicher des Rechners abgeleitet und nicht auf der Festplatte gespeichert.

Unter Windows 9x/ME werden unter dem Hauptschlüssel HKU die Benutzernamen der einzelnen Nutzer angezeigt, unter Windows 2000/XP werden anstelle der Benutzernamen Security IDs (SIDs) verwendet. Der Unterschlüssel *.Default* enthält das Profil für einen neu anzulegenden Nutzer. Ist bei Windows 9x/ME keine Benutzeranmeldung aktiviert, so wird das Default-Profil verwendet [Wor99][Wal01].

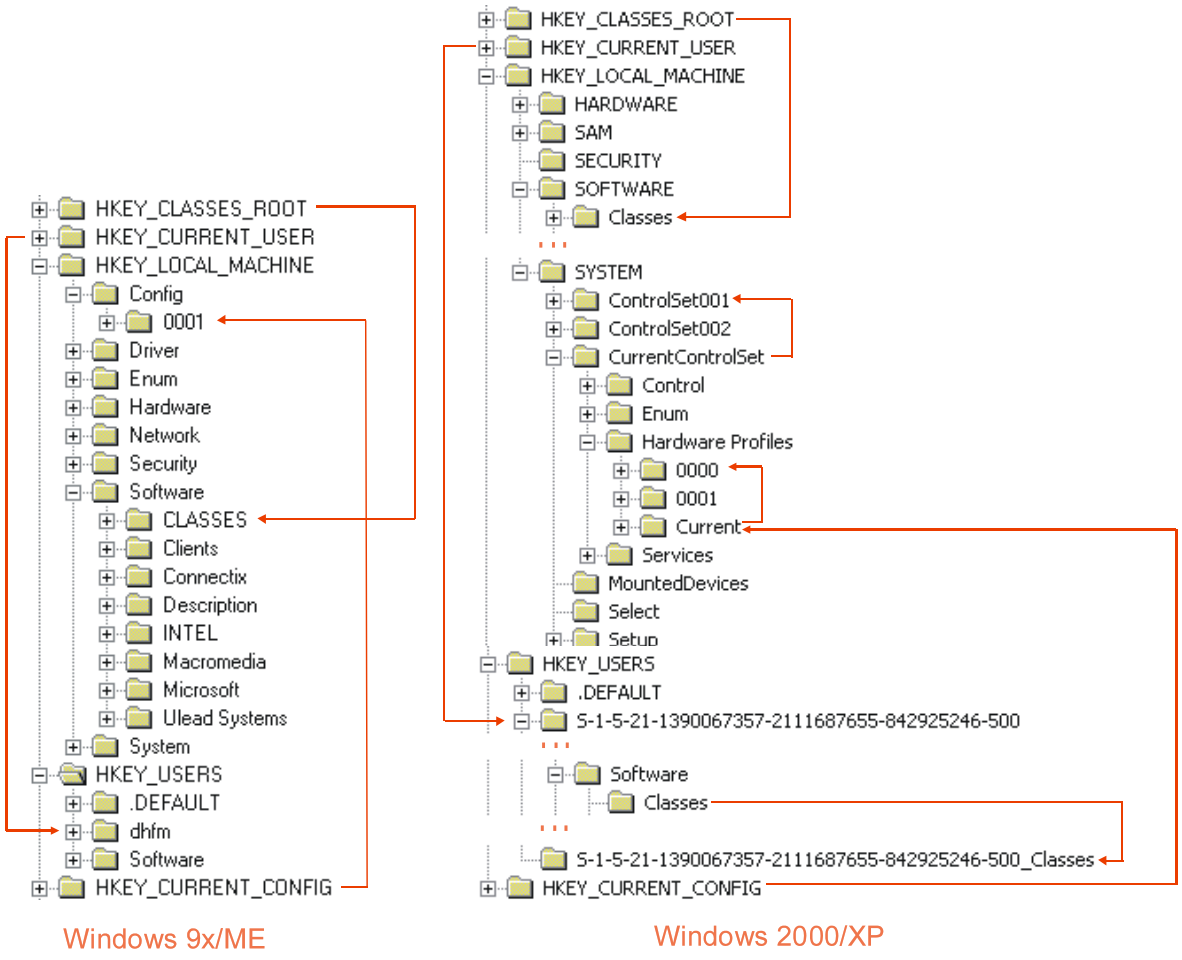


Abbildung 2 : graphische Darstellung des Anlegens von Links (Verweisen) nach dem Starten vom bzw. dem Anmelden am Windows 9x/ME- und 2000/XP-System im Vergleich

1.1.4. Die Datentypen der Registry

Werte, die in der Registry gespeichert sind, können in verschiedenen Datentypen gespeichert werden. Grundlegend wird zwischen folgenden Datentypen unterschieden:

- Strings
- Zahlen
- Binärwerte

Diese Typen gliedern sich wiederum in verschiedene Untergruppen, welche in Tabelle 2 dargestellt sind [Rob00][Wal01].

Datentyp	Beschreibung
REG_NONE	Eintrag ohne Datentyp
REG_BINARY*	Binäre Daten
REG_DWORD*	Double Word (32 Bit)
REG_DWORD_ BIG_ENDIAN	Double Word; das niederwertige Byte wird zuerst gespeichert
REG_DWORD_ LITTLE_ENDIAN	Double Word; das höherwertige Byte wird zuerst gespeichert
REG_QWORD	Quadruple Word (64 Bit)
REG_QWORD_ BIG_ENDIAN	Quadruple Word; das niederwertige Byte wird zuerst gespeichert
REG_QWORD_ LITTLE_ENDIAN	Quadruple Word; das höherwertige Byte wird zuerst gespeichert
REG_SZ*	einfacher String im Klartext
REG_MULTI_SZ	Stringliste, durch #0 getrennte Strings
REG_EXPAND_SZ	String, der %Variablen% enthalten kann, wie z.B. %Systemroot%
REG_FULL_RESOURCE_ DESCRIPTOR	vollständige Beschreibung einer Ressource; enthält z.B. belegte Interrupts, DMA-Blöcke und I/O-Ports Wird nicht lesbar als binärer Block in der Registry gespeichert
REG_RESOURCE_LIST	Liste von Ressourcen
REG_RESOURCE_ REQUIREMENTS_LIST	Liste von Vorraussetzungen bzgl. der Ressourcen
REG_LINK	Verweis auf eine andere Stelle der Registry

Tabelle 2 : Zusammenstellung der Datentypen der Registry

* diese Datentypen sind unter Windows 9x/ME und 2000/XP verfügbar, alle anderen nur unter Windows 2000/XP

1.1.5. Bearbeitungswerkzeuge für die Registry

1.1.5.1. Der Registrierungseditor

Zur Bearbeitung der Registry stellt Microsoft den Registrierungseditor *RegEdit* zur Verfügung. Über *Start/Ausführen -> RegEdit* kann dieser aufgerufen werden. In Abbildung 1 ist ein Screenshot des Registrierungseditors unter Windows 2000 abgebildet. Wie in dieser Abbildung zu erkennen ist, werden die Schlüssel hierarchisch im Explorer-Stil dargestellt. Auf der rechten Seite sind die dem jeweils markierten Schlüssel zugeordneten Einträge und ihre Werte abgebildet. Mit Hilfe des Registrierungseditors können Werte, Einträge und Schlüssel gelöscht, hinzugefügt und bearbeitet werden. Des Weiteren können Teile oder die gesamte Registry in Reg-Dateien exportiert bzw. aus ihnen importiert werden. Reg-Dateien speichern die exportierten Teile im Klartext ab, so dass sie mit einem Texteditor bearbeitet werden können.
Der Registrierungseditor bietet auch die Möglichkeit, Zugriff auf die Registry eines anderen, über das Netzwerk erreichbaren, Rechners zu erlangen. Auf diesem Rechner muss jedoch der Remote-Registry-Dienst aktiviert sein. Zusätzlich muss unter Windows 9x/ME eine Zugriffsbeschränkung auf Benutzerebene aktiviert sein. Des Weiteren muss derjenige, der auf die Registry zugreifen will, einen Account auf diesem Rechner haben. Die Informationen werden jedoch im Klartext übertragen.
Unter Windows 2000 steht zusätzlich noch der Registrierungseditor *RegEdt32* zur Verfügung, welcher in Abbildung 3 dargestellt ist.

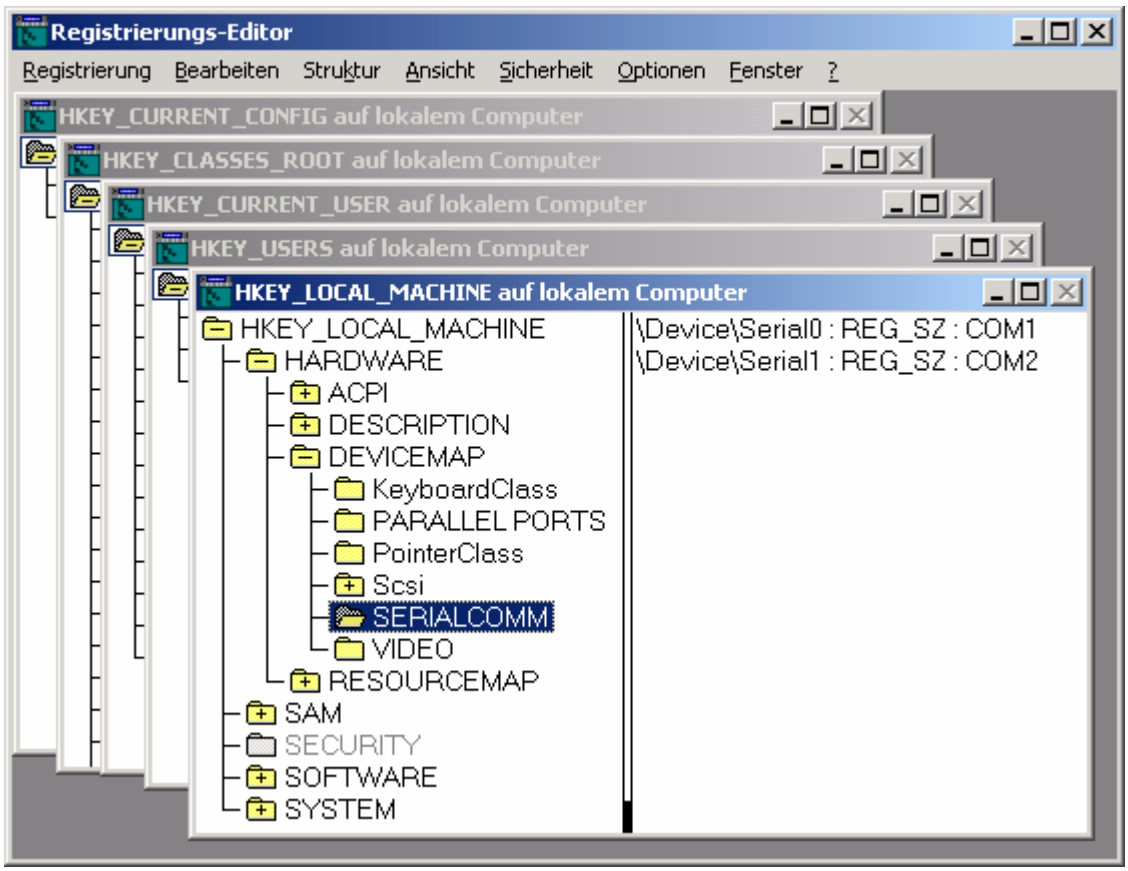


Abbildung 3 : Screenshot des Registrierungseditors RegEdt32

Dieser Editor stellt jeden Hauptschlüssel in einem eigenen Fenster dar. Eintrag, Datentyp und Wert werden, jeweils durch einen Doppelpunkt getrennt, auf der rechten Seite dargestellt. Das wichtigste Feature von *RegEdt32* ist die Möglichkeit Zugriffsbeschränkungen auf jeden Schlüssel zu setzen, wie es bei NTFS auf Dateiebene möglich ist. Diese Berechtigungen können wie auch bei NTFS auf Benutzer- oder Gruppenebene gesetzt werden. Weiterhin ist es möglich eine Zugriffsprotokollierung auf beliebige Schlüssel zu aktivieren, um Aktionen bestimmter Nutzer zu überwachen. Für die Zugriffsbeschränkung stehen folgende Optionen zu Auswahl:

- Wert abfragen
- Wert festlegen
- Unterschlüssel erstellen
- Unterschlüssel auflisten
- Benachrichtigungen (Setzen und Abfragen der Zugriffsprotokollierung)
- Verknüpfung erstellen (Erstellen von Verweisen)

- Löschen
- DAC schreiben (Berechtigungen ändern)
- Besitzer festlegen
- Lesezugriff (enthält Wert abfragen und Unterschlüssel auflisten)

Unter Windows XP wurde die Funktionalität, die *RegEdt32* bietet, in *RegEdit* integriert [Bor98][Rob00][Wal01].

1.1.5.2. Zugriff über API

Mit Hilfe der Windows-API ist es möglich, Zugriff auf die Registry aus einer Applikation heraus zu erhalten. Die Methoden der Windows-API sind meist in DLL-Dateien gekapselt. Die Zugriffsmethoden für die Registry befinden sich in der DLL *advapi32.dll*. Um einfachen Zugriff auf diese Methoden bzw. die DLL zu bekommen, wurden die Methodenaufrufe in der Unit *Windows* (unter Delphi. C, C++) exportiert, so dass der Programmierer sie transparent nutzen kann.
Unter Delphi wurde eine Klasse *TRegistry* implementiert, die diese Methoden nochmals kapselt, um dem Programmierer einen noch einfacheren Zugriff zu gewähren. Jedoch werden durch diese Klasse nicht alle Methoden, die die Windows-API für den Registry-Zugriff bietet, bereitgestellt [Rob00].

1.1.6. Die Dateien der Registry

1.1.6.1. Windows 9x/ME

Wie schon in Abschnitt 1.1.3 beschrieben, werden die Hauptschlüssel HKU und HKLM als Master-HKEYs bezeichnet, da die anderen Hauptschlüssel, bis auf HKDD, in Bereiche der Master-HKEYs zeigen. Diese beiden Master-HKEYs werden getrennt in zwei Dateien auf der Festplatte gespeichert. Rechnerspezifische Informationen, die in HKLM gespeichert werden, werden in der Datei *System.dat*, welche sich im Windowsordner befindet, abgelegt. Benutzerspezifische Daten des Hauptschlüssels HKU werden für jeden Nutzer im Verzeichnis *Windows\Profiles\<Nutzername>* in der Datei *User.dat* gespeichert. Das Default-Benutzerprofil, das die Grundeinstellungen für jeden neu anzulegenden Nutzer enthält, wird in der Datei *User.dat* im Windowsordner gespeichert. Durch die Trennung von rechnerspezifischen und nutzerspezifischen Daten ist es möglich, dass die nutzerspezifischen Daten dem Nutzer auf jedem Rechner zur Verfügung stehen, auf dem er sich anmeldet, wobei die Datei *User.dat* auf einem Netzlaufwerk gespeichert wird, auf das der jeweilige Rechner Zugriff hat [Bor98].
Um den Bootvorgang bei Windows ME zu beschleunigen, wird bei dieser Windowsversion der Hauptschlüssel HKCR, also *HKLM\Software\Classes*, in einer Extradatei Namens *Classes.dat* im Windowsordner gespeichert. Diese wird erst nach dem Starten des Betriebssystems geladen [Hor02].

1.1.6.2. Windows 2000/XP

Die Dateien, in denen Teilbereiche der Registry gespeichert werden, werden als Registry Hives bezeichnet. In Abbildung 4 sind die Dateipositionen der Registry Hives dargestellt. Wie in dieser Abbildung zu erkennen ist, arbeiten Windows 2000 und XP mit Partitionspositionen anstatt mit Laufwerksbuchstaben beim Angeben der Dateipositionen. Somit werden bei Änderungen der Laufwerksbuchstaben Dateien, dessen Pfad in der Registry gespeichert ist, problemlos gefunden.

Name	Wert
(Standard)	(Wert nicht gesetzt)
\REGISTRY\MACHINE\HARDWARE	
\REGISTRY\MACHINE\SAM	\Device\HarddiskVolume2\WINNT\system32\config\SAM
\REGISTRY\MACHINE\SECURITY	\Device\HarddiskVolume2\WINNT\system32\config\SECURITY
\REGISTRY\MACHINE\SOFTWARE	\Device\HarddiskVolume2\WINNT\system32\config\software
\REGISTRY\MACHINE\SYSTEM	\Device\HarddiskVolume2\WINNT\system32\config\system
\REGISTRY\USER\.DEFAULT	\Device\HarddiskVolume2\WINNT\system32\config\default
\REGISTRY\USER\S-1-5-21-1547161642-152049171-1957994488-1000	\Device\HarddiskVolume2\Dokumente und Einstellungen\dhfm\NTUSER.DAT
\REGISTRY\USER\S-1-5-21-1547161642-152049171-1957994488-1000_Classes	\Device\HarddiskVolume2\Dokumente und Einstellungen\dhfm\ ,Lokale Einstellungen\Anwendungsdaten\Microsoft\Windows\UsrClass.dat

Abbildung 4 : Auflistung der Dateipositionen der Registry Hives

Die in Abbildung 4 dargestellten Einträge *\Registry\Machine\...* beziehen sich auf HKLM und *\Registry\User\...* auf HKU.

Der Registry Hive *HKLM\Hardware* enthält keinen Wert, da hier dynamische Daten über den Rechner gespeichert sind, die nur im Arbeitsspeicher gehalten werden. Unter dem Schlüssel *HKLM\Sam* wird die Datenbank des Security Accounts Managers (SAM) abgelegt, welche Informationen über lokale Benutzer und Gruppen enthält. *HKLM\Security* enthält ebenfalls sicherheitsrelevante Einstellungen, wie Zugriffsbeschränkungen für Benutzer und Gruppen. Informationen und Einstellungen zu den installierten Applikationen werden unter dem Schlüssel *HKLM\Software*, rechnerspezifische Einstellungen und Informationen unter dem Schlüssel *HKLM\System* gespeichert. Wie schon in Abschnitt 1.1.3 erwähnt, werden lokale Benutzer nicht mit ihrem Benutzernamen, sondern über so genannte Security IDs gespeichert, die so ausgelegt sind, dass sie global einmalig und eindeutig auftreten. Eine Ausnahme bildet das Default-Nutzerprofil, da bei diesem speziellen Nutzerprofil keine Security ID, sondern der Name *.Default* verwendet wird. Dieses Profil liefert die Standardeinstellungen für jeden neu angelegten Nutzer. Standardmäßig werden die Dateien von HKLM und des Default-Nutzerprofils in Ordner *%Systemroot%\system32\config* gespeichert.

Benutzerprofile angelegter Nutzer werden in der Datei *NTUser.dat*, benutzerspezifische Dateiverknüpfungen, die unter dem Schlüssel *HKU\<Security_ID_des_Users>\Software\Classes* zu finden sind, in der Datei *UsrClass.dat*, jeweils für jeden Nutzer, gespeichert. Diese Dateien werden ursprünglich im Ordner *Dokumente und Einstellungen\<Nutzername>* im Falle von *NTUser.dat* und *Dokumente und Einstellungen\<Nutzername>\Lokale*

Einstellungen\Anwendungsdaten\Microsoft\Windows in Falle von *UsrClass.dat* gespeichert.
Somit ist bei der Registry von Windows2000/XP eine Trennung zwischen Hardware, Software, Sicherheit und Benutzern auf Dateiebene zu erkennen [Wal01][Rob00].

1.1.7. Der Bootvorgang von Windows

Am Beispiel von Windows 2000 wird in diesem Abschnitt erklärt, wie das Zusammenspiel der Registry mit dem Betriebssystem funktioniert.
Beim Starten von Windows 2000 wird ein Programm Namens *ntdetect.com* gestartet, welches den Computer nach vorhandener Hardware untersucht, zu denen folgende Objekte zählen:

- Anzahl, Art und Typ der Prozessoren
- Arbeitsspeicher
- I/O-Ports
- Bus-Typen (PCI, ISA etc.)
- Plug and Play Geräte
- Geräte, die auf diesen System-Bussen gefunden wurden

Diese Daten werden nach Erkennung im Arbeitsspeicher abgelegt und auf die Unterschlüssel von *HKLM\Hardware* abgebildet. Somit haben der Windows 2000-Kernel und Gerätetreiber Zugriff auf diese Daten. Diese Daten können zu diesem Zeitpunkt nicht auf der Festplatte gespeichert werden, da sie zu diesem Zeitpunkt erst erkannt wird und die benötigen Treiber noch nicht geladen sind.
Nach dem Erkennen der Hardware wird der Windows 2000-Kernel geladen, welcher unter anderem die Aufgabe besitzt, das Laden der richtigen Treiber für jedes gefundene Gerät zu steuern. Grundlage bieten die unter dem Schlüssel *HKLM\Hardware* gespeicherten Informationen. Jeder Treiber belegt Systemressourcen wie Interrupts, Port-Adressen und gegebenfalls DMA-Blöcke. Da diese Informationen ebenfalls unter dem Schlüssel *HKLM\Hardware* gespeichert werden, sollen durch diese zentrale Erfassung der Einstellungen und Ressourcen-Belegung Konflikte verhindert werden [Wal01][Rob00].

1.1.8. Die Speicherung von Ressourcen in der Registry

1.1.8.1. HKEY_USERS (HKU)

1.1.8.1.1. Windows 9x/ME

Laut [Mic98] sind unter diesem Hauptschlüssel alle Benutzerprofile der lokal angelegten Benutzer gespeichert. Tatsächlich befinden sich unter HKU nur das Default-Profil und das Profil des aktuell angemeldeten Nutzers. Ist keine Zugriffssteuerung auf Benutzerebene aktiviert oder wird die Anmeldung umgangen, so bezieht der Nutzer seine Einstellungen aus dem Default-Profil, das sonst die Grundeinstellungen für neue anzulegende Nutzer enthält [Bor98][Wor99].
Eine Liste aller lokal angelegten Nutzer befindet sich samt Information über den Speicherpfad unter dem Schlüssel *HKLM\Software\Microsoft\Windows\CurrentVersion\ProfileList* [*].

1.1.8.1.2. Windows 2000/XP

Unter Windows 2000/XP werden unter dem Hauptschlüssel HKU alle lokal angelegten Nutzerprofile aufgelistet. Jedoch werden nicht die Nutzernamen, sondern die Security IDs der jeweiligen Nutzer angezeigt, die global eindeutig sind, damit jedes Benutzerprofil eindeutig dem jeweiligen Nutzer zugeordnet werden kann, auch bei Verwendung der gleichen Login-Namen. Da bei Windows 2000/XP die benutzerspezifischen Dateiverknüpfungen in einer eigenen Datei gespeichert sind, werden diese als eigener Unterschlüssel von HKU in der Form *HKU\<SecurityID>_Classes* aufgeführt und ein Link auf *HKU\<SecurityID>\Software\Classes* gelegt [Rob00][Wal01].
In Abbildung 5 werden die Einträge von HKU für Windows 9x/ME und Windows 2000/XP gegenübergestellt.

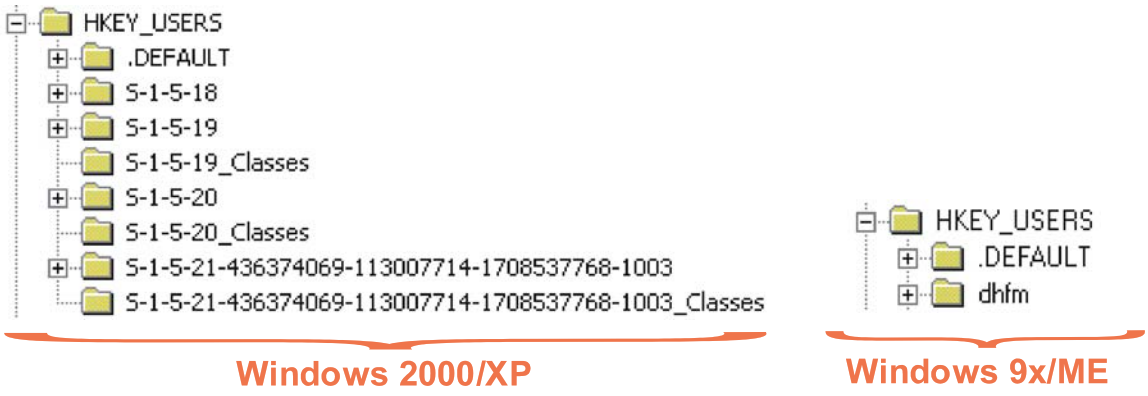


Abbildung 5 : graphische Darstellung der Aufsplittung des Hauptschlüssels HKU in Unterschlüssel

1.1.8.2. HKEY_CURRENT_USER (HKCU)

Dieser Hauptschlüssel ist ein Link auf das Nutzerprofil des aktuell angemeldeten Nutzers, welches unter HKU gespeichert ist. Somit muss eine Applikation nicht ermitteln, welcher Nutzer aktuell am System angemeldet ist.
Unter HKCU werden benutzerspezifische Einstellungen am Windowssystem und anderen installierten Applikationen gespeichert. Somit ist mit Hilfe der Registry ein Mehrbenutzerbetrieb unter Windows möglich. Tabelle 3 zeigt einen Überblick über die wichtigsten Schlüssel von HKCU. Diese sind bei Windows 9x/ME und Windows 2000/XP vorhanden, werden aber zum Teil erst bei Änderungen der Nutzer an den Standardeinstellungen angelegt [Bor98][Rob00] [Wor99].

Schlüssel	Inhalt
AppEvents\	Jeder Vorgang (event) kann unter Windows mit einem Geräusch hinterlegt werden. Diese Informationen werden unter diesem Schlüssel zusammengefasst. Das Programm Akustische Signale der Systemsteuerung liest die Informationen aus diesem Schlüssel.
EventLabels	enthält die Bezeichnungen der Vorgänge (events)
Schemes	enthält die Pfade und Dateinamen der abzuspielenden Dateien
Control Panel\	Unter diesem Schlüssel werden Einstellungen des Desktops, die in der Systemsteuerung angepasst werden können, zusammengefasst.

Tabelle 3 : Übersicht über die wichtigsten Schlüssel von HKCU unter Windows 9x/ME/2000/XP (Fortsetzung Seite 21)

Accessibility	Dieser Schlüssel enthält Verfügbarkeitsoptionen für z.B. Anzeige, Maus und Tastatur.
Appearance	Unter diesem Unterschlüssel werden Informationen über die unter dem Programm Anzeige wählbaren Darstellungsschemas gespeichert.
Colors	Trifft der Nutzer eigene Einstellungen an den Darstellungsschemas, die von Standardwerten abweichen, werden diese unter diesem Unterschlüssel zusammengefasst.
Cursors	Unter diesem Schlüssel werden die wählbaren Mauszeigerschemas gespeichert.
Desktop	Dieser Schlüssel enthält Einstellungen der Register Bildschirmschoner und Hintergrund des Programms Anzeige.
PowerCfg	Unter diesem Schlüssel werden Informationen über die Einstellungen der Energiesparfunktion von Monitor und Festplatten gespeichert.
keyboard layout\	Informationen über das Tastatursprachenlayout werden unter diesem Schlüssel gespeichert.
Network\	Verbindet sich der Rechner mit einer Netzwerkfreigabe eines anderen Rechners, so werden Informationen über diese Verbindungen unter diesem Schlüssel gespeichert.
Persistent	Wurde eine Freigabe mit einem lokalen Laufwerksbuchstaben verknüpft und soll diese Verbindung auch bei einem Neustart des Systems wieder hergestellt werden, so werden Informationen über diese Verbindung unter diesem Schlüssel gespeichert.
Recent	Informationen über gegenwärtige Verbindungen werden unter diesem Schlüssel zusammengefasst.
Software\	Benutzerspezifische Konfigurationen der installierten Applikationen werden unter diesem Schlüssel gespeichert, wobei jede Applikation ihren eigenen Schlüssel besitzt.

Tabelle 3 : Übersicht über die wichtigsten Schlüssel von HKCU unter Windows 9x/ME/2000/XP (fortgesetzt von 20)

Bei Windows 2000/XP gibt es drei weitere wichtige Schlüssel unter HKCU [Rob00]:

- Console (Einstellungen der Eingabeaufforderung)
- Environment (benutzerspezifische Umgebungsvariablen)
- Printers (Drucker, die der Nutzer verwenden kann, sowie der Standarddrucker)

1.1.8.3. HKEY_LOCAL_MACHINE (HKLM)

Dieser Hauptschlüssel enthält unter anderem alle Daten über die Hardware des Rechners, die Treiber- und Systemkonfiguration, die Einstellungen für das Netzwerk und Sicherheitsvorgaben.

1.1.8.3.1. Windows 9x/ME

Der Hauptschlüssel HKLM besitzt die folgenden Unterschlüssel, die nachstehend erläutert werden:

- Config
- Enum
- Hardware
- Network
- Security
- Software
- System

Config

Dieser Schlüssel enthält die verschiedenen Hardwarekonfigurationen, die für den lokalen Rechner verfügbar sind. Die Unterschlüssel haben die Form 000x, wobei x für 1,2,3 usw. steht, was den wählbaren Hardwarekonfigurationen entspricht. Der Hauptschlüssel HKCC wird mit der aktiven Konfiguration verlinkt, so dass die Erläuterung der Unterschlüssel im Abschnitt von HKCC vollzogen wird.
Welche verschiedenen Konfigurationen auf dem Rechner zur Verfügung stehen, werden im Programm System der Systemsteuerung unter dem Register Hardwareprofile zusammengefasst [Wor99].

Enum

Dieser Schlüssel enthält Informationen über die am System angeschlossene Hardware. Die einzelnen Komponenten werden nach Art der Hardware zusammengefasst. Jeder Art wird durch einen Unterschlüssel von *Enum* repräsentiert, was in Tabelle 4 dargestellt ist [Bor98][Wor99].

Unterschlüssel von Enum	Hardwareart
BIOS	Plug and Play-Komponenten vom BIOS, wie z.B. Zeitgeber und Controller
EISA	ISA Plug and Play-Bus-Geräte
ESDI	ESDI-Geräte, wie z.B. Festplatten
FLOP	Diskettenlaufwerke
HTREE	reserviert für zukünftige Nutzung
INFRARED	Infrarotgeräte
ISAPNP	ISA-Geräte
LPTENUM	Plug and Play-Drucker
MF	Multifunktionsgeräte, wie IDE-Controller
MONITOR	Monitore
Network	Netzwerkparameter wie Clients, Dienste und Protokolle
PCI	am PCI-Bus angeschlossene Geräte
PCMCIA	PCMCIA-Karten
Root	Geräte, die nicht zur Kategorie Plug and Play gehören
SCSI	SCSI-Geräte und CDROM-Laufwerke
SERENUM	an einem seriellen Anschluss angeschlossene Geräte
TAPECONTROLLER	Bandlaufwerke
USB	USB-Geräte

Tabelle 4 : Zusammenstellung der Unterschlüssel von Enum

Die Schlüssel enthalten weitere Unterschlüssel, unter denen Hardwarekomponenten gleichen Typs, wie z.B. Drucker und Festplatten, zusammengefasst werden. Unter diesen Unterschlüsseln befinden sich die Beschreibungen der einzelnen Hardwarekomponenten. In Abbildung 6 sind der Registry-Pfad sowie die Einträge und Werte einer Komponente dargestellt.

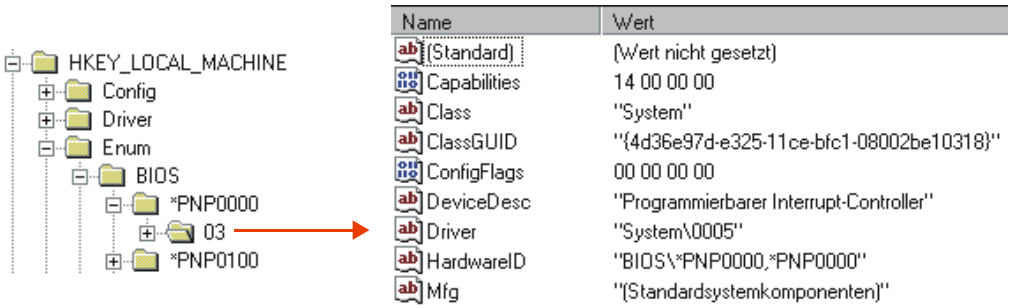


Abbildung 6 : graphische Darstellung der Einträge für eine Hardwarekomponente unter dem Schlüssel Enum

Die wichtigsten Einträge, die nicht jede Komponente besitzt, sind [Mic02][*]:

- BootConfig (spezifiziert die benutzten Ressourcen, wie Interrupts und I/O-Ports der Komponente)
- Capabilities (Informationen zur Leistungsfähigkeit)
- Class (Klasse, der die Komponente angehört)
- ClassGUID (eindeutige Bezeichnung der Klasse, da *Class* nicht eindeutig sein könnte)
- ConfigFlags (Konfigurationseinstellungen)
- DeviceDesc (Beschreibung der Komponente)
- Driver (Pfad zur Treiberbeschreibung relativ zu *HKLM\System\CurrentControlSet\Services\Class*)
- Mfg (Name des Herstellers)

Die Werte, die die Einträge *Capabilities* und *ConfigFlags* annehmen können, sind in Tabelle 5 bzw. Tabelle 6 dargestellt [Mic02].

Capabilities		
Konstante	**Wert**	**Bedeutung**
CM_DEVCAP_LOCKSUPPORTED	0x00000001	Komponente kann gesperrt werden, um das Auswerfen von Medien zu verhindern
CM_DEVCAP_EJECTSUPPORTED	0x00000002	Komponente unterstützt softwaregesteuerten Auswurf von Medien
CM_DEVCAP_REMOVABLE	0x00000004	Komponente kann dynamisch vom System entfernt werden
CM_DEVCAP_DOCKDEVICE	0x00000008	spezifiziert ob die Komponente ein angekoppeltes Peripheriegerät ist
CM_DEVCAP_UNIQUEID	0x00000010	Komponente unterstützt systemweit eindeutige Identifikationen. Dieses Bit ist nicht gesetzt, wenn die Komponente eindeutige Identifikationen nur innerhalb des Busses unterstützt
CM_DEVCAP_SILENTINSTALL	0x00000020	Der Gerätemanager soll alle Nachrichten bei der Installation der Komponente unterdrücken.
CM_DEVCAP_RAWDEVICEOK	0x00000040	Komponente kann durch den Treiber des darunterliegenden Busses angesteuert werden. Wird als Raw-Modus bezeichnet
CM_DEVCAP_SURPRISEREMOVALOK	0x00000080	Komponente kann sicher, ohne ein extra Programm zum Auswerfen, entfernt werden
CM_DEVCAP_HARDWAREDISABLED	0x00000100	Die Komponente ist deaktiviert. Übergeordnete Bustreiber oder Busfiltertreiber setzen dieses Flag, wenn kein Treiber für diese Komponente gefunden wurde
CM_DEVCAP_NONDYNAMIC	0x00000200	reserviert für die Zukunft

Tabelle 5 : Zusammenstellung der Werte, die der Eintrag *Capabilities* annehmen kann

ConfigFlags		
Konstante	**Wert**	**Bedeutung**
CONFIGFLAG_DISABLED	0x00000001	Komponente deaktiviert
CONFIGFLAG_REMOVED	0x00000002	Komponente entfernt
CONFIGFLAG_MANUAL_INSTALL	0x00000004	Komponente wurde manuell installiert
CONFIGFLAG_IGNORE_BOOT_LC	0x00000008	Bootkonfiguration überspringen
CONFIGFLAG_NET_BOOT	0x00000010	Komponente laden, wenn Systemstart mit Netzwerkunterstützung aktiviert wurde
CONFIGFLAG_REINSTALL	0x00000020	Installation der Komponente wiederholen
CONFIGFLAG_FAILEDINSTALL	0x00000040	Installation fehlgeschlagen
CONFIGFLAG_CANTSTOPACHILD	0x00000080	Eine einzelne untergeordnete Komponente kann nicht gestoppt oder entfernt werden.
CONFIGFLAG_OKREMOVEROM	0x00000100	Komponente kann auch bei Schreibschutz entfernt werden
CONFIGFLAG_NOREMOVEEXIT	0x00000200	Beim Beenden nicht entfernen
CONFIGFLAG_FINISH_INSTALL	0x00000400	Installation, der im Raw-Modus laufende Komponente, vervollständigen
CONFIGFLAG_NEEDS_FORCED_CONFIG	0x00000800	Diese Komponente benötigt eine zwingende Konfiguration.
CONFIGFLAG_NETBOOT_CARD	0x00001000	Die Komponente ist eine Netzwerkkarte, über die über das Netzwerk gebootet werden kann.
CONFIGFLAG_PARTIAL_LOG_CONF	0x00002000	Diese Komponente besitzt eine partielle Log-Konfiguration.

Tabelle 6 : Zusammenstellung der Werte, die der Eintrag *ConfigFlags* annehmen kann

Hardware

Dieser Schlüssel soll laut [Mic98] Informationen über serielle Anschlüsse und Modems, die mit dem Programm Hyper Terminal verwendet werden, enthalten. Jedoch enthält der Unterschlüssel *Description* nur Bezeichnungen der Prozessoren, des Co-Prozessors, sowie der Multifunktionsgeräte [Bor98] [Wor99].

Network

Wurde auf dem System ein DFÜ-Netzwerk oder andere Netzwerkkonfigurationen eingerichtet, enthält dieser Schlüssel den Unterschlüssel *Logon*, welcher Informationen über den angemeldeten Nutzer beinhaltet [Wor99].

Security

Dieser Schlüssel enthält Daten über den Remote Access-Dienst, wenn dieser installiert wurde [Wor99].

Software

Dieser Schlüssel enthält Informationen und Konfigurationen über die installierten Applikationen, die für alle Benutzer gelten, unabhängig von ihren persönlichen Konfigurationen.
Er enthält unter anderem auch die Unterschlüssel *Microsoft*, welcher Informationen und Konfigurationen zu allen installierten Microsoft-Applikationen, zu Windows und zu denen im Umfang mit Windows mitgelieferten Applikationen birgt. Unter dem Schlüssel *Microsoft* befindet sich der Unterschlüssel *Windows\CurrentVersion*, unter dem Informationen zum Windows-Betriebssystem zusammengefasst sind. Die Einträge dieses Unterschlüssels enthalten allgemeine Informationen zu Windows, wie Windowsversion, Registrierungs-

Informationen, Installationspfade und sogar den Registrierungs-Schlüssel von Windows im Klartext.
Des Weiteren befinden sich unter dem Schlüssel *Windows\CurrentVersion* unter anderem auch die folgenden Unterschlüssel [Wor99][*]:

- Internet Settings (Informationen über allgemeine Interneteinstellungen)
- ProfileList (enthält Informationen über alle lokal angelegten Nutzer (Nutzername, Pfad zu *User.dat*))
- Run (enthält Einträge der Programme, die bei jeden Systemstart nach der Anmeldung geladen werden sollen)
- RunOnce (enthält Einträge der Programme, die nur beim nächsten Systemstart nach der Anmeldung geladen werden sollen; danach werden diese Einträge gelöscht)
- Sevices bzw. ServicesOnce (analog zu Run bzw. RunOnce, mit dem Unterschied, dass es sich hier um Dienste handelt)
- Uninstall (Unter diesem Schlüssel sind Informationen über alle Programme gespeichert, die über das Programm Software der Systemsteuerung deinstalliert werden können.)

Sind auf dem Computer Netzwerkfreigaben eingerichtet, so befinden sich Informationen über diese Freigaben unter dem Schlüssel *Windows\CurrentVersion\Network\LanMan* [*]. Jede Netzwerkfreigabe besitzt unter diesem Schlüssel einen eigenen Unterschlüssel, der den Namen der Freigabe trägt. Jede Freigabe enthält folgende Einträge [*]:

- Path (lokaler Pfad der Freigabe)
- Remark (Beschreibung der Freigabe)
- Parm1enc (verschlüsseltes Schreibschutzkennwort)
- Parm2enc (verschlüsseltes Leseschutzkennwort)
- Flags (Art des Zugriffsschutzes)

Der Eintrag *Flags* nimmt einen der folgenden dezimalen Werte an [*]:

- 257 (schreibgeschützt)
- 258 (lese/schreibgeschützt)
- 259 (Zugriff abhängig vom Kennwort)
- 401 (keine Änderung an den Standardwerten)

System

Dieser Schlüssel enthält Daten, die beschreiben, wie das Windowssystem konfiguriert ist. Unter dem Schlüssel *System* befindet sich der Unterschlüssel *CurrentControlSet*, der selber die beiden Unterschlüssel *Control* (Steuerung) und *Services* (Dienste) enthält. Die wichtigsten Unterschlüssel sind für *Control* in Tabelle 7 und für *Services* in Tabelle 8 zusammengefasst [Wor99][Bor98][*].

Unterschlüssel	Bedeutung
ComputerName	Aktueller Computername des Rechners
IDConfigDB	enthält die Namen der aktiven und aller anderen Hardwarekonfigurationen, die unter dem Register Hardwarekonfiguration des Programms System der Systemsteuerung wählbar sind
Keyboard layouts	enthält alle Tastaturlayouts
Mediaresources	Dieser Unterschlüssel fasst Informationen der Multimediageräte, die unter dem Register Geräte des Programms Sounds und Multimedia der Systemsteuerung abgebildet sind, zusammen.
PrefStats	enthält Namen und Beschreibungen der Komponenten, zu denen im Systemmonitor Leistungsstatistiken betrachtet werden können
SessionManager	enthält eine Auflistung der DLLs, die beim Starten von Windows in den Speicher geladen werden, sowie eine Auflistung der Programme und DLLs, die eventuell Probleme verursachen könnten oder Probleme haben könnten, einwandfrei unter Windows zu laufen

Tabelle 7 : Zusammenstellung der wichtigsten Unterschlüssel von *Control* (Windows 9x/ME)

Unterschlüssel	Bedeutung
Class	enthält die eindeutigen Klassenbezeichnungen (ClassGUID) der Komponentenklassen, sowie Informationen zu den Treibern der installierten Geräte, wie z.B. INF-Datei und Treiberdatum
Class\NetTrans	enthält Informationen über die TCP/IP-Konfiguration der Netzwerkinterfaces, wie z.B. IP-Adresse, DNS, Gateway
VxD	enthält Informationen über die virtuellen Gerätetreiber

Tabelle 8 : Zusammenstellung der wichtigsten Unterschlüssel von *Services* (Windows 9x/ME)

1.1.8.3.2. Windows 2000/XP

Unter Windows 2000 und XP befinden sich fünf Schlüssel unter HKLM:

- Hardware
- Sam
- Security
- Software
- System

Jeder dieser Schlüssel ist ein Registry Hive, so dass diese Schlüssel auch physisch auf Dateiebene getrennt sind.

Hardware

Der Inhalt dieses Schlüssels wird aus dem RAM des Systems abgeleitet und nicht auf der Festplatte gespeichert. Unter diesem Schlüssel befinden sich Informationen zur Hardware des Rechners, auf dem Windows gestartet wird.
So befinden sich z.B. Informationen zum Prozessor (inklusive MHz-Zahl), Co-Prozessor, Multifunktionsadapter und die Versionen des Grafikkarken- und System-BIOS unter dem Unterschlüssel *Description*. Diese Informationen werden auf x86-Maschinen durch das Programm *ntdetect.com* beim

Systemstart unter diesem Schlüssel abgelegt [Rob00][Wal01].
Der Unterschlüssel *Devicemap* enthält Unterschlüssel, die für verschiedene Geräteklassen stehen, wie z.B. *Video* (steht für Anzeigegeräte). Unter diesen Schlüsseln werden die einzelnen der Klasse zugehörigen Geräte als Einträge gespeichert. Diese Einträge enthalten als Werte Links den zugehörigen Treibern bzw. Konfigurationsdaten. Der Eintrag *\Device\Video0* steht z.B. für das erste Anzeigegerät und enthält als Wert den Link zum Treiber oder zu Konfigurationsdaten, unter denen detaillierte Informationen zum Gerät, wie die Größe des Speichers oder den RAMDAC-Typ, zu finden sind [*].
Der Unterschlüssel *Resourcemap* des Schlüssels Hardware enthält Informationen über Hardware-Ressourcen, die von den einzelnen Geräten belegt werden [Rob00]. Diese Einträge tragen jedoch Bezeichnungen, die nicht auf den Namen des Gerätes schließen lassen.

Sam

Dieser Schlüssel ist ein Link auf *HKLM\Security\Sam* und enthält die Datenbank des Security Accounts Managers (SAM), in der lokale Anwender und Gruppen abgelegt sind.
Standardmäßig hat nur das System Zugriff auf diese Daten. Der Administrator hat jedoch die Möglichkeit, die Berechtigungen auf diesem Schlüssel zu ändern. Die Daten, die sich unter diesem Schlüssel befinden, sind nicht im Klartext gespeichert. Sie sind verschlüsselt, so dass auch ein Administrator nicht die Möglichkeit hat, z.B. an Passwörter der einzelnen Nutzer zu gelangen [Rob00][Wal01].

Security

Unter diesem Schlüssel sind neben der SAM-Datenbank auch Richtlinien (Policies) oder die Zugriffseinstellungen der Registry-Objekte gespeichert. Wie auch beim Schlüssel *Sam* kann auf diese ebenfalls verschlüsselten Daten standardmäßig nicht zugegriffen werden [Rob00][Wal01].

Software

Wie auch unter Windows 9x und ME werden unter diesem Schlüssel Informationen und Konfigurationen zu den installierten Applikationen und Windows gespeichert. Damit Applikationen eine einheitliche Umgebung zum Speichern ihrer Konfigurationen vorfinden, wurde in Windows 2000/XP die Architektur dieses Schlüssels von Windows 9x/ME übernommen. Da unter Windows 2000 und XP Dienste mit Hilfe des Dienstkontroll-Managers verwaltet werden, befinden sich unter dem Schlüssel Software keine Informationen zu Diensten. Des Weiteren werden Informationen über Netzwerkfreigaben nicht unter diesem Schlüssel gespeichert.

Neben dem Schlüssel *Microsoft\Windows* gibt es bei Windows 2000/XP den Schlüssel *Microsoft\Windows NT*, der spezifische Informationen und Konfigurationen zu Windows 2000/XP enthält. So werden Windowsversion, Registrierungs-Informationen und Installationspfade nicht wie bei Windows 9x/ME in den Einträgen von *Microsoft\Windows\CurrentVersion*, sondern in den Einträgen des Schlüssels *Microsoft\Windows NT\CurrentVersion* gespeichert. Der Unterschlüssel *ProfileList*, der Informationen zu den Nutzern enthält, wird ebenso unter *Windows NT* und nicht unter *Windows* gespeichert.

Nachfolgend werden die wichtigsten Unterschlüssel von *Microsoft\Windows NT\CurrentVersion* beschrieben [Rob00].

Informationen über die im System installierten Audio-, Videocodecs, Mediensteuer- und Videoaufnahmegeräte , die unter dem Register Hardware des Programms Sounds und Multimedia der Systemsteuerung zu finden sind, werden unter den Schlüsseln *Drivers32* (Audio-/Videocodecs, Videoaufnahmegeräte) und *MCI32* (Mediensteuergeräte) zusammengefasst. Die Einträge der Audiocodecs haben die Form *acm.x*, die der Videcodecs *vidc.x* und die der Videoaufnahmegeräte *msvideo*. Existiert eine Beschreibung für die hinter den Einträgen gespeicherten Dateien, so wird diese unter dem Schlüssel *drivers.desc* gespeichert [*].

Des Weiteren existieren noch die folgenden Unterschlüssel [*]:

- NetworkCards (enthält die Bezeichnungen der installierten Netzwerkkarten und der zugehörigen Treiber)
- Perflib (Dieser Unterschlüssel enthält die Bezeichnungen aller Komponenten, zu denen Leistungsstatistiken mit Hilfe des Performancemonitors betrachtet werden können. Der Unterschlüssel *009* enthält die Bezeichnungen in englischer Sprache. Daneben existiert noch ein Unterschlüssel, der die Bezeichnungen in der Sprache der installierten Windowsversion enthält. Im Falle einer deutschen Version, trägt dieser Unterordner die Bezeichnung *007*.)
- Print (enthält Informationen zu den installierten Druckern)

System

Dieser Schlüssel enthält Daten, die für den Systemstart benötigt werden. Dies sind unter anderem Informationen und Konfigurationen der Gerätetreiber und Dienste. Wie auch unter Windows 9x/ME werden diese Einstellungen unter einem ControlSet[1] unter dem Schlüssel *HKLM\System*\, mit dem Unterschied, dass bei Windows 98/ME nur der Unterschlüssel *CurrentControlSet* existiert, zusammengefasst. Unter Windows 2000/XP existieren zusätzlich die Unterschlüssel *ControlSetXXX*, wobei *XXX* für 001,002, etc steht, und verschiedenen Konfigurationseinstellungen entsprechen. Somit ist es möglich bei heiklen Änderungen an diesen Einstellungen, auf die vorherige Konfiguration zurückzugreifen, um die vorherigen Einstellungen wiederherstellen zu können. Es werden mindestens zwei ControlSets gespeichert: das letzte geladene und das letzte, was erfolgreich gestartet wurde. Der Unterschlüssel *CurrentControlSet* ist ein Link auf das aktive ControlSet.

[1] ControlSet steht allgemein für ControlSetXXX bzw. CurrentControlSet

Der Schlüssel *System* besitzt neben diesen Unterschlüsseln noch die folgenden [Rob00][Wal01]:

- MountedDevices (wird von NTFS benutzt, um die Laufwerksbezeichnungen mit den internen Laufwerksidentifikationsnummern zu verbinden)
- Setup (enthält Einstellungen, die vom Windows 2000/XP-Setupprogramm benutzt werden, um herauszufinden welche Installationsphasen abgeschlossen sind und an welcher Stelle sich die Installation befindet)
- Select (enthält Informationen, welches ControlSet das aktive (Current), das standardmäßige (Default), das letzte, das erfolgreich gestartet ist (LastKnownGood), sowie das, das als letztes nicht erfolgreich gestartet ist (Failed))

Ein ControlSet enthält vier Unterschlüssel, die nachstehend näher erläutert werden [Rob00][Wal01]:

- Control (enthält Informationen über die Systemkonfiguration)
- Enum (enthält Informationen über die Konfiguration der Hardwaregeräte)
- HardwareProfiles (enthält Informationen über die verschiedenen Hardware-Profile)
- Services (enthält Informationen über die Konfiguration der Dienste)

Ein graphischer Überblick über diese Schlüsselhierarchie ist in Abbildung 2 zu finden.

System\ControlSet\Control

In Tabelle 9 sind die wichtigsten Unterschlüssel von *Control* zusammengefasst [Rob00][Wal01][*].

Unterschlüssel	Bedeutung
Class	Unter diesem Schlüssel werden die Klassenbezeichnungen (ClassGUID) der Komponentenklassen, sowie Informationen zu den Treibern der installierten Geräte, wie z.B. INF-Datei und Treiberdatum zusammengefasst. Entspricht bei Windows 9x/ME dem Schlüssel *HKLM\System\CurrentControlSet\Services\Class*.
Computername	enthält wie auch bei Windows 98/ME den Namen des Computers
hivelist	enthält Informationen über die Registry Hives samt Pfadangaben (vgl. 1.1.6.2)
IDConfigDB	enthält wie bei Windows 98/ME den Namen der aktiven und aller anderen Hardwarekonfigurationen
Network	enthält Informationen über die installierten Netzwerkadapter, samt TCP/IP-Konfiguration, wie z.B. IP-Adresse, DNS, Gateway
Print	enthält wie auch der Schlüssel *HKLM\Software\Print* Informationen über die installierten Drucker
ProductOptions	enthält Informationen über die genaue Windowsversion (Professional, Server, Terminal Server etc.)

Tabelle 9 : Die wichtigsten Unterschlüssel von *Control* (Windows 2000/XP)

System\ControlSet\Enum

Dieser Schlüssel enthält Informationen über die am System angeschlossene Hardware. Die Hardwareinformationen werden auf die gleiche Weise unter dem Schlüssel *HKLM\Enum* unter Windows 9x/ME gespeichert (siehe Punkt 1.1.8.3.1 Abschnitt *Enum*). Da die meisten Gerätetreiber unter Windows 2000/XP als Dienst implementiert sind, enthalten die Schlüssel der Hardwaregeräte, zusätzlich unter Windows 2000/XP, den Eintrag *Service*, der den Namen des Dienstes enthält, zu dem Informationen unter dem Schlüssel *HKLM\System\ControlSet\Services* zu finden sind. Des Weiteren besitzen die Hardwarekomponenten, die am System angeschlossen sind, den Unterschlüssel *Control*. Unter *HKLM\System\ControlSet\Enum* werden die Informationen über Hardwarekomponenten so lange gespeichert, bis sie mit dem Programm Hardware der Systemsteuerung oder dem Gerätemanager deinstalliert wurden [Rob00][Wal01].

System\ControlSet\Hardware Profiles

Dieser Schlüssel enthält die verschiedenen Hardwarekonfigurationen, die für den lokalen Rechner verfügbar sind. Dieser Schlüssel entspricht dem Schlüssel *HKLM\Config* unter Windows 9x/ME (siehe Punkt 1.1.8.3.1 Abschnitt *Config*). Im Gegensatz zu Windows 9x/ME befindet sich unter diesem Schlüssel noch ein Unterschlüssel *Current*, der ein Link auf die aktive Hardwarekonfiguration ist. Demzufolge gibt es zwei Wege, auf die aktive Hardwarekonfiguration zu gelangen: über HKCC und *HKLM\System\ControlSet\Hardware Profiles/Current* [Rob00].

System\ControlSet\Services

Windows 2000/XP-Komponenten, die als Dienste implementiert wurden, wie z.B. Gerätetreiber und Server oder Applikationen, die im Hintergrund laufen, hinterlegen ihre Konfigurationen unter diesem Schlüssel. In Abbildung 7 sind die Informationen, die zu einem Dienst gespeichert werden können, graphisch dargestellt [Rob00].

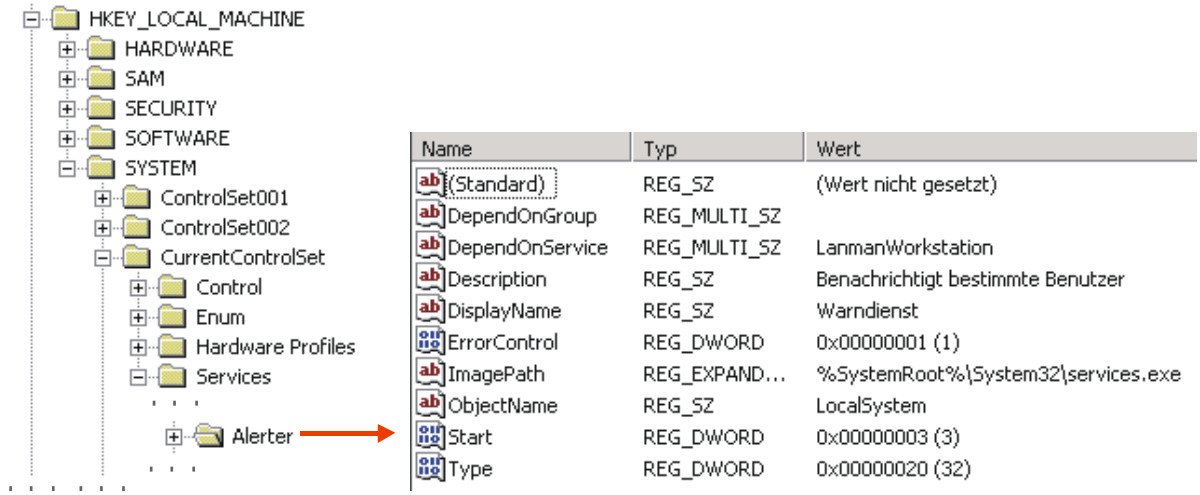


Abbildung 7 : graphische Darstellung der Einträge und Werte eines Dienstes unter dem Schlüssel *Services* (Windows 2000/XP)

Die Einträge, die ein Dienst besitzen kann sind [Rob00]:

- DependOnGroup (Ein Dienst aus dieser Gruppe sollte gestartet sein, bevor der Dienst, der diesen Eintrag besitzt, gestartet wird.)
- DependOnService (Dienste, die hier genannt werden, müssen vor dem Start des Dienstes gestartet sein.)
- Description (Beschreibung zum Dienst)
- DisplayName (Name des Dienstes, der unter dem Punkt Dienste der Management Console angezeigt werden soll)
- ErrorControl (spezifiziert, was bei einem fehlerhaften Start des Dienstes passieren soll)
- Group (enthält den Namen der Gruppe, der der Dienst angehört)
- ImagePath (enthält Pfad und Namen der ausführbaren Datei des Dienstes)
- ObjectName (Name des Accounts unter dem der Dienst laufen soll)
- Start (spezifiziert wann der Dienst gestartet werden soll)
- Tag (spezifiziert die Startreihenfolge in einer Gruppe. Der Dienst mit der niedrigsten Nummer startet zuerst)
- Type (spezifiziert den Typ des Dienstes oder Treibers)

Der Eintrag *ErrorControl* besitzt einen der folgenden dezimalen Werte:

- 0 (Fehler ignorieren und keine Fehlermeldungen ausgeben)
- 1 (Fehlernachricht ausgeben, aber mit dem Startprozess fortfahren)
- 2 (Startprozess fortsetzen, wenn das aktive ControlSet das letzte erfolgreich gestartete ControlSet ist; wenn nicht, soll das als letztes erfolgreich gestartete geladen werden)
- 3 (Das aktive ControlSet soll als fehlerhaft markiert werden. Wenn das letzte erfolgreich gestartete ControlSet das aktive ist, dann soll ein Diagnoseprogramm gestartet werden. Ansonsten soll das letzte erfolgreich gestartete geladen werden.)

Der Eintrag *Type* enthält einen der folgenden hexadezimalen Werte:

- 001 (Kernel-Gerätetreiber)
- 002 (Kernel-Gerätetreiber, der einen Dateisystemtreiber implementiert)
- 004 (Sammlung von Argumenten, die von Netzwerkadaptern benutzt werden)
- 008 (Dateisystem-Treiber)
- 010 (Win32-Dienst, der als Standalone-Prozess laufen soll)
- 020 (Win32-Dienst, der Adressen mit anderen Diensten des gleichen Typs teilen kann)
- 110 (Win32-Dienst, der als Standalone-Prozess laufen soll und mit Benutzern interagieren kann)
- 120 (Win32-Dienst, der Adressen mit anderen Diensten des gleichen Typs teilen kann und mit Benutzern interagieren kann)

Der Eintrag *Start* besitzt einen der folgenden dezimalen Werte:

- 0 (Boot - Treiber soll als erstes geladen werden, da er nötig ist, um den Startdatenträger nutzbar zu machen)
- 1 (System - Dienst sollte durch das I/O-System geladen werden, wenn der Kernel geladen wurde)
- 2 (Autoload - Dienst soll automatisch geladen und gestartet werden)
- 3 (Manual - Dienst soll geladen werden, aber der Nutzer muss ihn manuell starten)
- 4 (Disabled - Dienst soll geladen werden, aber soll nicht durch das System oder den Benutzer gestartet werden)

Jeder Dienst kann folgende Unterschlüssel besitzen:

- Linkage (Netzwerkadapter können an Protokolle oder Dienste gebunden sein. Die Pfade der angebundenen Komponenten werden unter diesem Unterschlüssel gespeichert. Informationen über deaktivierte Bindungen werden unter *Linkage**Disabled* gespeichert.)

- Performance (Dienste, dessen Aktivitäten über den Performance Monitor überwacht werden können, speichern unter diesem Unterschlüssel Informationen, wie die Zählernummer, über die der Dienst bzw. die Aktivität des Dienstes identifiziert werden kann.)
- Security (Dieser Unterschlüssel enthält Informationen zur Zugriffsberechtigung für den Dienst.)
- Enum (enthält den Pfad der Hardwarekomponenten, die der Dienst steuert oder mit denen er interagiert)

Lokale Netzwerkfreigaben werden unter Windows 2000/XP unter dem Dienst *lanmanserver* und nicht wie bei Windows 9x/ME unter dem Schlüssel *HKLM\Software\Microsoft\Windows\CurrentVersion\Network\LanMan*, gespeichert. Jede Netzwerkfreigabe wird in einem eigenen Eintrag, der den Namen der Freigabe trägt, als REG_MULTI_SZ gespeichert. Sicherheitsrelevante Informationen zu jeder Freigabe werden unter dem Unterschlüssel *Security*, dessen Einträge ebenfalls den Namen der Freigabe tragen, als verschlüsselte binäre Sequenz gespeichert [*].

1.1.8.4. HKEY_CLASSES_ROOT (HKCR)

Wie in Abschnitt 1.1.3 erwähnt wurde, ist dieser Hauptschlüssel ein Verweis auf *HKLM\Software\Classes* und dient einem einfacheren Zugriff auf diesen Bereich, der sich sonst zwei Ebenen unter HKLM abspielen würde. Weiterhin sichert er die Abwärtskompatibilität zu Windows 3.1-Applikationen.
Die Unterschlüssel von HKCR können in 3 Bereiche aufgeteilt werden:

- Dateitypen
- zugeordnete Applikationen
- Klassenidentifizierungen für COM, OLE und ActiveX

Die Dateitypen werden durch ihre Dateierweiterung klassifiziert. Ihnen ist ein Punkt vorangestellt, z.B. *.txt*. Die Kennung der diesem Dateityp zugeordneten Applikation ist als Wert dieses Schlüssels gespeichert. Der blaue Pfeil in

Abbildung 8 verdeutlicht eine solche Bindung. Die roten Pfeile deuten auf die Einträge bzw. Werte der am Pfeilanfang stehenden Schlüssel.

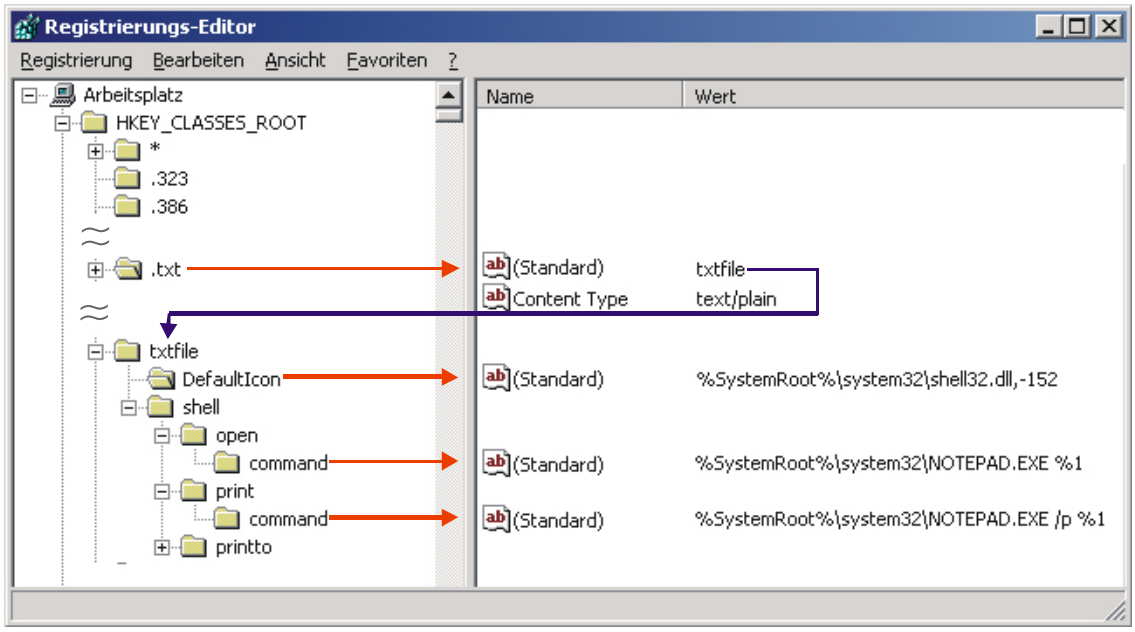


Abbildung 8 : graphische Darstellung der Beziehung zwischen Dateierweiterung und zugeordneter Applikation

Die der Dateierweiterung zugeordnete Applikation, in Abbildung 8 *txtfile*, besitzt die Unterschlüssel *DefaultIcon*, in dem der Verweis zum Icon gespeichert ist, das im Explorer angezeigt werden soll, sowie den Unterschlüssel *shell*, der weitere Unterschlüssel enthält, unter denen unter anderem gespeichert ist, welche Anweisung bei einem Doppelklick auf Dateien dieser Dateierweiterung (*open\command*) oder bei Klicken auf Drucken im Kontextmenü dieser Dateien (*print\command*) ausgeführt werden soll.

Zum aktiven Austauschen von Daten zwischen Applikationen stehen unter Windows OLE, COM und ActiveX zur Verfügung. Um sämtliche Programm-Module bzw. Softwareobjekte eindeutig identifizieren zu können, müssen sie systemweit eindeutig zuordenbar sein. Jede Softwareinstanz ist mit einer eigenen Class ID (CLSID) gekennzeichnet, die einmalig und eindeutig ist. Diese CLSIDs sind in dem Unterschlüssel *CLSID* des Hauptschlüssel HKCR gespeichert. Die wichtigsten Unterschlüssel jeder Class ID sind *InprocServer32* und *InprocServer*, welche die DLL spezifizieren, die die für die Erzeugung und Bearbeitung der jeweiligen Objekte notwendigen Methoden enthält [Bor98] [Rob00][Wal01].

1.1.8.5. HKEY_CURRENT_CONFIG (HKCC)

Dieser Hauptschlüssel ist wie in Abbildung 2 dargestellt, ein Link auf das aktive Hardwareprofil (vgl.1.1.8.3.1 Punkt *Config* und 1.1.8.3.2 Punkt *System*\ *ControlSet**Hardware Profiles*). Werden neben dem standardmäßig vom System angelegten Hardwareprofil eigene angelegt, so kann beim Systemstart entschieden werden, welches Profil verwendet werden soll.
Unter HKCC werden Hardwareeinstellungen gespeichert, die in jedem Hardwareprofil anders sein können. Hauptsächlich werden unter diesem Hauptschlüssel Änderungen an der Bildschirmauflösung, sowie der Verwendbarkeit der installierten Hardware gespeichert. Änderungen an der Verwendbarkeit werden in einem Flag festgehalten, das als Eintrag unter dem Schlüssel der jeweiligen Hardwarekomponente gespeichert wird. Die Bezeichnung dieses Eintrages ist *CSConfigFlags* und kann die in Tabelle 10 dargestellten Werte annehmen [Mic02][*].

Konstante	Wert	Bedeutung
CSCONFIGFLAG_DISABLED	0x00000001	Hardwarekomponente deaktiviert
CSCONFIGFLAG_DO_NOT_CREATE	0x00000002	Eintrag für Hardwarekomponente soll nicht angelegt werden
CSCONFIGFLAG_DO_NOT_START	0x00000004	Treiber der Hardwarekomponente soll nicht geladen werden

Tabelle 10 : Zusammenstellung der Werte des Eintrages *CSConfigFlags*

1.1.8.6. HKEY_DYN_DATA (HKDD)

Dieser Hauptschlüssel ist nur unter Windows 9x/ME verfügbar, wird dynamisch vom RAM des Systems abgeleitet und nie auf der Festplatte gespeichert.
Unter diesem Hauptschlüssel befinden sich zwei Unterschlüssel [Bor98]:

- Config Manager (enthält Informationen über den Hardwarebaum)
- PerfStats (enthält Informationen über Leistungsstatistiken)

Config Manager

Dieser Schlüssel beschreibt die installierten Hardwarekomponenten nach ihren Anschlüssen. Im Gerätemanager besteht die Möglichkeit, die installierte Hardware nach Typen oder nach Anschlüssen anzeigen zu lassen. Bei einer Anzeige nach Typen werden die Informationen aus dem Schlüssel *HKLM\Enum* gelesen und nach Hardwareklassen (z.B. CD-ROM, Grafikkarten) zusammengefasst dargestellt. Bei einer Anzeige nach Anschlüssen werden die Informationen aus dem Schlüssel *Config Manager* gelesen. Da bei der Anzeige nach Hardwareklassen nur zwei Ebenen (Klasse und Hardwarekomponente) und bei der Anzeige nach Anschlüssen mehrere Ebenen dargestellt werden können (z.B. Plug and Play BIOS -> PCI-Bus -> IDE Controller -> Festplatte), wird diese Darstellung auch als Hardwarebaum bezeichnet [*].
Die einzelnen Hardwarekomponenten werden unter dem Unterschlüssel *Config Manager\Enum* durch eine acht Zeichen umfassende Hexadezimalnummer (z.B. C29A8A10) identifiziert. Jede Hardwarekomponente besitzt einen eigenen Unterschlüssel unter *Config Manager\Enum*, der den Namen der Identifikationsnummer der Hardwarekomponente trägt. Die einzelnen Hardwarekomponenten sind als Knoten eines Baumes mit Identifikationsnummer des Vaters, Sohnes und Nachbarn gespeichert. Jede Hardwarekomponente enthält folgende Einträge [*]:

- HardWareKey (Link zur Beschreibung der Hardwarekomponente relativ zu *HKLM\Enum*)
- Child (Identifikationsnummer des Sohnes)
- Parent (Identifikationsnummer des Vaters)
- Sibling (Identifikationsnummer des Nachbarn)
- Problem (enthält den Gerätestatus der Komponente)

Der Eintrag *Problem*, also der Gerätestatus der Komponente, wird auch im Gerätemanager angezeigt. Sobald er ungleich Null ist, wird das Icon der Komponente im Gerätemanager mit einem Ausrufezeichen dargestellt [*]. Welchen Wert der Eintrag *Problem* annehmen kann, wird in Abschnitt 1.2.1 beschrieben.

PerfStats

Dieser Schlüssel liefert Informationen zu den Leistungsstatistiken, die im Systemmonitor (Performance Monitor) von Windows angezeigt werden können [Bor98]. Die Erklärungen und Bezeichnungen der darstellbaren Elemente werden aus dem Schlüssel *HKLM\System\ControlSet\Control\PerfStats* gelesen (vgl. 1.1.8.3.1 Punkt *System*) [*].

1.1.8.7. HKEY_PERFORMANCE_DATA (HKPD)

Dieser Hauptschlüssel enthält, wie der Hauptschlüssel HKEY_DYN_DATA unter Windows 9x/ME, dynamische Daten aus dem RAM des Systems, die nicht auf der Festplatte gespeichert werden. HKPD existiert nur unter Windows 2000/XP. Unter HKPD werden im Gegensatz zu HKDD nur Leistungsstatistiken gespeichert [Rob00][Wal01].
Jedoch wird er in keinem der Registrierungseditoren (*RegEdit* bzw. *RegEdt32*) von Windows angezeigt. Die Funktionen der Registry-API, über die alle Unterschlüssel und Einträge aufgelistet werden können, geben diese für HKPD nicht zurück. Es kann also nur direkt auf Schlüssel oder Einträge zugegriffen werden.
Der Performance Monitor von Windows liest über diesem Hauptschlüssel die Leistungsstatistiken aus. Die Erklärungen und Bezeichnungen der Leistungsstatistiken, die zu bestimmten Elementen überwacht werden können, werden unter den Schlüsseln *HKLM\Software\Microsoft\Windows NT\CurrentVersion\Perflib* (vgl. 1.1.8.3.2 Punkt *Software*) und *HKLM\System\ControlSet\Services\<Name des Dienstes>\Performance* (vgl. 1.1.8.3.2 Punkt *System\ControlSet\Services*) gespeichert [*].

1.2. Windows-DLLs

DLLs (Dynamic Link Libraries) sind kompilierte Modulbibliotheken mit offen gelegten Schnittstellen (Methoden, Ressourcen), die sich durch andere Applikationen mit DLL-Schnittstelle zur Laufzeit einbinden lassen. DLLs bieten zum einen den Vorteil, dass sie in Applikationen verwendet werden können, die in verschiedenen Programmiersprachen (Delphi, C++, Visual Basic etc.) entwickelt wurden und zum anderen kann eine DLL zur gleichen Zeit von mehreren Applikationen verwendet werden, wobei der Programmcode der DLL nur einmal im Arbeitsspeicher abgelegt wird.
Die Methoden der Windows-API sind in DLLs gekapselt und sind somit in Programmiersprachen, in denen DLLs eingebunden werden können, nutzbar [Dob00][Kos00].
Mit Hilfe der Windows-API lassen sich, wie auch mit Hilfe der Registry, Ressourcen-Analysen durchführen, die teilweise auf die Registry zugreifen, teilweise aber auch auf den Kernel von Windows.
In diesem Abschnitt werden einige DLLs, in denen Methoden der Windows API gekapselt wurden, kurz erläutert. Dabei stehen die Ressourcen-Analysen im Vordergrund, deren Quelle nicht die Registry darstellt und die in der Konzeption und Implementierung verwendet werden.
Welche Parameter die jeweiligen Methoden erwarten und zurückliefern kann aus [Mic02][Mic02a] entnommen werden.

1.2.1. setupapi.dll

Diese DLL enthält Methoden für die Treiberentwicklung unter Windows. Diese Methoden interagieren größtenteils mit der Registry von Windows.
Unter Windows 9x/ME wird der Gerätestatus der Hardwarekomponenten unter dem Hauptschlüssel HKDD gespeichert (vgl. 1.1.8.6). Somit kann ein eventueller Fehlercode direkt aus der Registry gelesen werden. Unter Windows 2000/XP kann der Gerätestatus nur mit Hilfe der Methoden der *setupapi.dll* ausgelesen werden. Dazu stehen die Funktionen *CM_Locate_DevNodeA* (liefert ein Handle auf eine Komponente) und *CM_Get_DevNode_Status* (liefert

mit Hilfe des Handle den Gerätestatus der Komponente) zur Verfügung. Der Fehlercode, der durch die Funktion *CM_Get_DevNode_Status* zurückgegeben wird, nimmt wie auch der Eintrag *Problem* unter HKDD bei Windows 9x/ME (vgl. 1.1.8.6), einen der in Tabelle 11 dargestellten Wert an, der auch im Gerätemanager dargestellt wird, wo im Falle eines Wertes ungleich Null im Icon der jeweiligen Komponente ein Ausrufezeichen angezeigt wird [Mic02][Com02] [Tri02][Hae02].

Konstante	Wert	Bedeutung
	0x00000000	kein Fehler
CM_PROB_NOT_CONFIGURED	0x00000001	Das System hat das Gerät nicht konfiguriert.
CM_PROB_DEVLOADER_FAILED	0x00000002	Der Gerätelader konnte den Gerätetreiber nicht laden.
CM_PROB_OUT_OF_MEMORY	0x00000003	Speicher reicht nicht aus um das Gerät zu laden
CM_PROB_ENTRY_IS_WRONG_TYPE	0x00000004	Die INF-Datei des Gerätes weist Fehler auf.
CM_PROB_LACKED_ARBITRATOR	0x00000005	Für den Ressourcentyp des Gerätes gibt es keinen Eintrag.
CM_PROB_BOOT_CONFIG_CONFLICT	0x00000006	Es besteht ein Ressourcenkonflikt.
CM_PROB_FAILED_FILTER	0x00000007	Das System konnte das Gerät nicht konfigurieren.
CM_PROB_DEVLOADER_NOT_FOUND	0x00000008	Ein Eintrag in der INF-Datei zeigt auf eine Datei, die nicht existiert.
CM_PROB_INVALID_DATA	0x00000009	Ein Eintrag in der Registry zu diesem Gerät weist Fehler auf.
CM_PROB_FAILED_START	0x0000000A	Das Gerät ist nicht vorhanden, funktioniert nicht oder es sind nicht alle Treiber installiert.
CM_PROB_LIAR	0x0000000B	Gerät defekt oder Fehler in der Konfiguration
CM_PROB_NORMAL_CONFLICT	0x0000000C	Es sind keine freien Ressourcen für das Gerät vorhanden.
CM_PROB_NOT_VERIFIED	0x0000000D	Das Gerät ist nicht vorhanden, funktioniert nicht oder es sind nicht alle Treiber installiert.
CM_PROB_NEED_RESTART	0x0000000E	Geräteproblem kann durch einen Neustart des Rechners behoben werden.
CM_PROB_REENUMERATION	0x0000000F	Dem Gerät wurde eine Ressource zugeteilt, die bereits von einem anderen Gerät beansprucht wird.
CM_PROB_PARTIAL_LOG_CONF	0x00000010	Es konnten nicht alle Geräte-ressourcen identifiziert werden.

Tabelle 11 : Zusammenstellung der Werte des Gerätestatus einer Komponente

(Fortsetzung auf Seite 47)

CM_PROB_UNKNOWN_RESOURCE	0x00000011	Die in der Treiberinformationsdatei angegebene Ressource für das untergeordnete Gerät wird vom übergeordneten Gerät nicht erkannt.
CM_PROB_REINSTALL	0x00000012	Das Gerät muss neu installiert werden.
CM_PROB_REGISTRY	0x00000013	Möglicherweise ist die Registry beschädigt.
CM_PROB_VXDLDR	0x00000014	Es konnte kein Treiber für das Gerät geladen werden.
CM_PROB_WILL_BE_REMOVED	0x00000015	Das Gerät wird entfernt.
CM_PROB_DISABLED	0x00000016	Das Gerät ist deaktiviert bzw. wurde nicht gestartet.
CM_PROB_DEVLOADER_NOT_READY	0x00000017	Das Gerät ist nicht vorhanden, funktioniert nicht oder es sind nicht alle Treiber installiert.
CM_PROB_DEVICE_NOT_THERE	0x00000018	Das Gerät wurde nicht gefunden.
CM_PROB_MOVED	0x00000019	Dieser Fehler kann nur nach dem ersten Neustart vom Windows-Setup, also nie im Gerätemanager, erscheinen.
CM_PROB_TOO_EARLY	0x0000001A	Das Gerät wird konfiguriert.
CM_PROB_NO_VALID_LOG_CONF	0x0000001B	Die Ressourcen für das Gerät können nicht bestimmt werden.
CM_PROB_FAILED_INSTALL	0x0000001C	Für dieses Gerät sind keine Treiber installiert.
CM_PROB_HARDWARE_DISABLED	0x0000001D	Das Gerät ist deaktiviert, da im BIOS keine Ressourcen angegeben sind.
CM_PROB_CANT_SHARE_IRQ	0x0000001E	Dieses Gerät greift auf eine IRQ-Ressource zu, die bereits von einem anderen Gerät verwendet wird und nicht gemeinsam genutzt werden kann.
CM_PROB_FAILED_AD	0x0000001F	Das Gerät funktioniert nicht richtig.
CM_PROB_DISABLED_SERVICE	0x00000020	Die Gerätetreiber konnten nicht installiert werden.
CM_PROB_TRANSLATION_FAILED	0x00000021	Das Gerät reagiert nicht auf den Treiber.

Tabelle 11 : Zusammenstellung der Werte des Gerätestatus einer Komponente

(fortgesetzt von Seite 46)

1.2.2. kernel32.dll

Aus dieser DLL werden in der Konzeption und Implementierung die in Tabelle 12 dargestellten Methoden verwendet [Mic02a].

Methode	Beschreibung
GlobalMemoryStatus	gibt den freien und belegten Arbeitsspeicher und Speicher der Auslagerungsdatei zurück
GetDriveType	ermittelt den Typ (Festplatte, Disketten-laufwerk etc.) eines Laufwerks
GetDiskFreeSpaceEx	ermittelt den freien und gesamten Speicher eines Laufwerks

Tabelle 12 : Zusammenstellung der verwendeten Methoden aus der DLL kernel32.dll

1.2.3. iphlpapi.dll

Diese DLL bietet Methoden, mit denen die Funktionalitäten der Programme ipconfig.exe, netstat.exe, route.exe, arp.exe nachgebildet werden können.
Die in Tabelle 13 dargestellten Methoden aus dieser DLL werden in der Konzeption und Implementierung verwendet [Pen01][Mic02a].

Methode	Beschreibung
GetIfEntry	liefert die maximale theoretische Übertra-gungsgeschwindigkeit der Verbindung einer Netzwerkkarte, sowie die MAC-Adresse
GetAdaptersInfo	liefert zu einem Adapter den Index, der z.B. für *GetIfEntry* benötigt wird
GetIpForwardTable	gibt die Routingtabelle zurück
GetIpAddrTable	liefert zu einem Index die IP-Adresse; wird von GetIpForwardTable benötigt, da auch Loopbackdevices enthalten sind

Tabelle 13 : Zusammenstellung der verwendeten Methoden aus der DLL iphlpapi.dll

1.2.4. user32.dll

Aus dieser DLL werden in der Konzeption und Implementierung die in Tabelle 14 dargestellten Methoden verwendet [Mic02a].

Methode	Beschreibung
EnumDisplayDevicesA	liefert alle grafischen Anzeigegeräte des Systems
EnumDisplaySettingsA	liefert alle unterstützten Auflösungen eines Anzeigegerätes

Tabelle 14 : Zusammenstellung der verwendeten Methoden aus der DLL user32.dll

1.3. Das System Management BIOS (SMBIOS)

Das BIOS (Basic Input Output System) bildet die Schnittstelle zwischen Hardware und Software. Das BIOS erfüllt z.B. folgende Funktionalitäten [Kof01][Ulb96]:

- Selbsttest und Initialisierung der Komponenten des Computersystems (POST[1]) beim Einschalten des Rechners
- Bereitstellung von Routinen zur Steuerung und zum Datenaustausch mit diesen Komponenten

Die Position des BIOS im Adressraum des Arbeitsspeichers ist standardisiert. Der BIOS-Code wird im F-Segment (*0F0000h*) eingeblendet. Die genaue Position in diesem Segment ist von der Größe des Codes abhängig. Der BIOS-Code muss aber bei *0FFFFFh* enden [Ulb96]. Informationen über gefundene Komponenten des Computersystems beim POST befinden sich ebenfalls in diesem Adressraum.
Der Eintrittspunkt der ersten Komponenten-Informationen im BIOS-Code kann aus der Eintrittspunkt-Struktur, die auszugsweise in Tabelle 15[Dis02] dargestellt ist, entnommen werden.

Offset	Name	Länge	Inhalt
00h	Anker-String	4 Bytes	**_SM_**, spezifiziert als 4 ASCII-Zeichen (5F 53 4D 5F)
…	…	…	…
06h	SMBIOS Haupt-Version	Byte	Hauptversion des SMBIOS
07h	SMBIOS Unter-Version	Byte	Unterversion des SMBIOS
…	…	…	…
10h	Zwischen-String	5 Bytes	**_DMI_**, spezifiziert als 5 ASCII-Zeichen (5F 44 4D 49 5F)
…	…	…	…
18h	Eintrittspunkt	DWORD	Eintrittpunkt für Komponenten-Informationen
1Ch	Anzahl der Komponenten-Informationen	WORD	Anzahl der Komponenten-Informationen
…	…	…	…

Tabelle 15 : Darstellung der Eintrittspunkt-Struktur

[1] POST (Power On Self Test)

Diese Struktur kann durch Durchsuchen des BIOS-Codes nach den ASCII-Strings *_SM_* und/oder *_DMI_* gefunden werden.
Abbildung 9 zeigt auszugsweise ein Beispiel einer Eintrittspunkt-Struktur. Die erste Spalte gibt die Position im Speicher und die zweite innerhalb der Eintrittspunkt-Struktur an. Spalte drei und vier zeigen den Inhalt im Hex- bzw. im ASCII-Format an der entsprechenden Adresse an. Die Beschreibungen hinter den geschweiften Klammern erklären die Bedeutung der dargestellten Informationen.

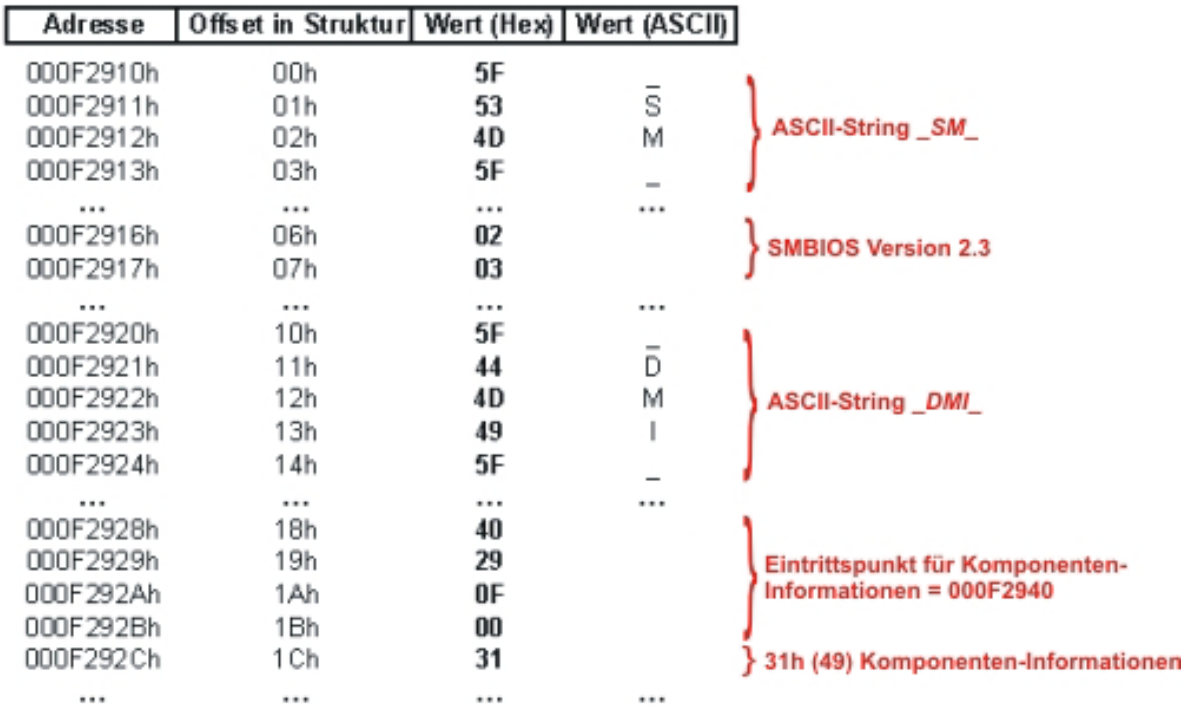


Abbildung 9 : Beispiel einer Eintrittspunkt-Struktur

Jede Komponenten-Information besitzt eine eigene Struktur, wobei die ersten vier Bytes jeder Struktur das gleiche Format aufweisen - der Header. Der Aufbau des Headers ist in Tabelle 16 dargestellt.

Offset	Name	Länge	Inhalt
00h	Typ	Byte	spezifiziert den Typ der Komponenten-Information
01h	Länge	Byte	spezifiziert die Länge der Komponenten-Information
02h	Handle	Word	spezifiziert den Handle der Komponenten-Information, der für SMBIOS-Funktionen verwendet wird

Tabelle 16 : Darstellung des Headers der Komponenten-Informationen

Besitzt eine Komponenten-Information Textstrings, so sind diese direkt hinter der Komponenten-Information gespeichert und durch *00h* voneinander getrennt. Die Komponenten-Informationen werden hintereinander gespeichert und sind durch *0000h* separiert.

Die Art der Komponenten-Informationen ist durch die Typ-Angabe im Header charakterisiert. Welche verschiedenen Typen im BIOS-Code gefunden werden können, kann der SMBIOS-Spezifikation [Dis02] entnommen werden. Erwähnt sei nur der Typ *7Fh* (*127*), der das Ende der Komponenten-Informationen spezifiziert.

In Abbildung 10 ist der Aufbau und der Inhalt der Komponenten-Information vom Typ *00h* (BIOS-Information) an einem Beispiel dargestellt.

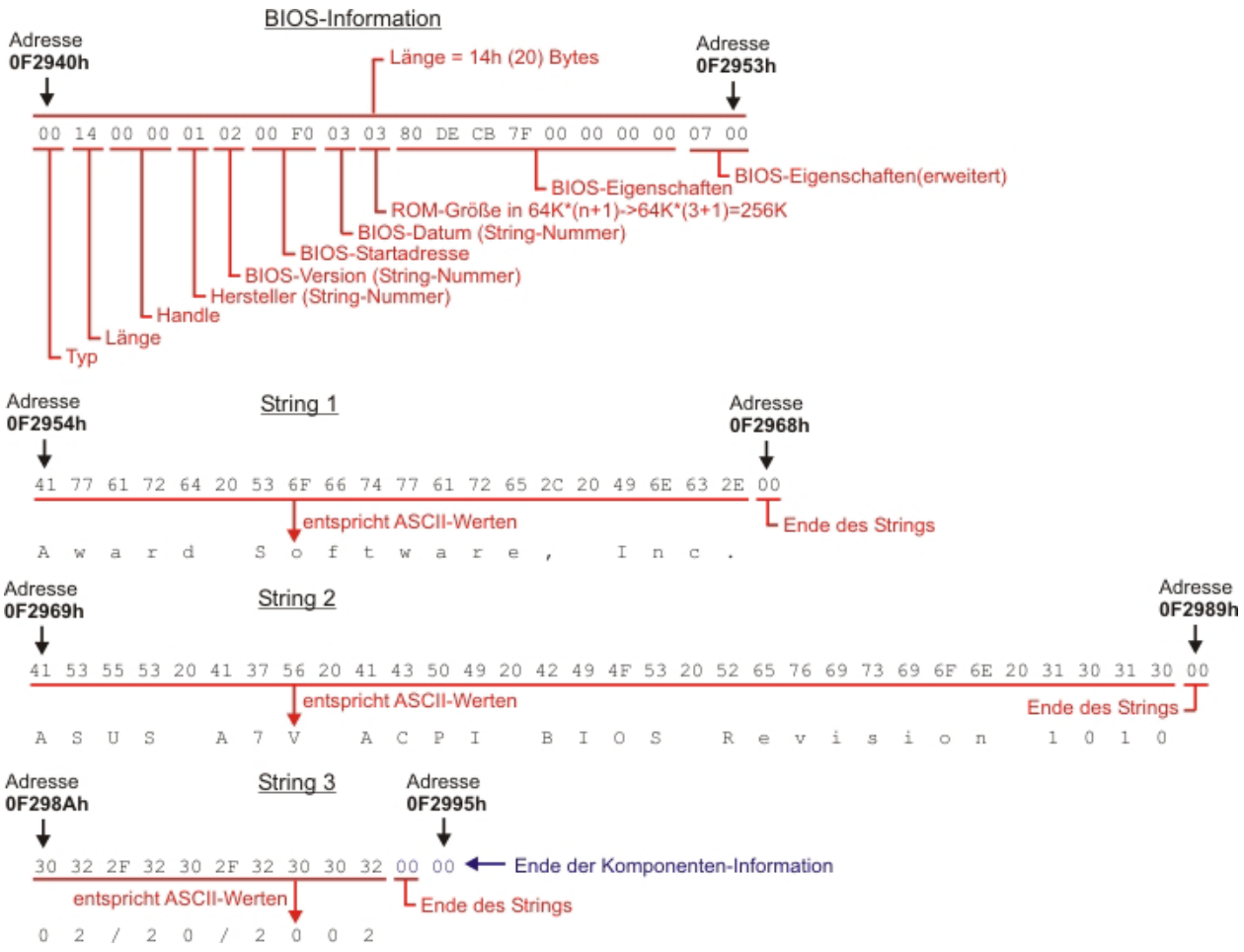


Abbildung 10 : Darstellung des Aufbaus und des Inhaltes der Komponenten-Information BIOS-Information an einem Beispiel

Abbildung 10 gliedert sich in zwei Abschnitte. Zum ersten ist dies die BIOS-Informations-Struktur und zum anderen die Strings der BIOS-Information, die an die Struktur angehangen sind. Wie aus der Abbildung zu entnehmen ist, ist die BIOS-Informationen wie folgt aufgeteilt:

- Adresse *0F2940h* - *0F2953h* : **BIOS-Informations-Struktur**
- Adresse *0F2954h* - *0F2967h* : **String 1**
- Adresse *0F2968h* - *0F2968h* : Ende String 1 (*00h*)
- Adresse *0F2969h* - *0F2988h* : **String 2**
- Adresse *0F2989h* - *0F2989h* : Ende String 2 (*00h*)
- Adresse *0F298Ah* - *0F2993h* : **String 3**
- Adresse *0F2994h* - *0F2994h* : Ende String 3 (*00h*)
- Adresse *0F2994h* - *0F2995h* : Ende der BIOS-Information (*0000h*)

Wie in der Abbildung dargestellt, werden in den Einträgen *Hersteller*, *BIOS-Version* und *BIOS-Datum* die entsprechenden String-Nummern gespeichert [Dis02]. Somit sind die Informationen über *Hersteller*, *BIOS-Version* bzw. *BIOS-Datum* in den Strings 1,2 bzw. 3 gespeichert.
Was sich hinter den *BIOS-Eigenschaften* verbirgt, kann der SMBIOS-Spezifikation [Dis02] entnommen werden.
Die anderen Komponenten-Informationen sind wie die BIOS-Information aufgebaut. Jedoch sind Größe und Einträge abhängig von der entsprechenden Komponenten-Information. Dies kann ebenfalls der Spezifikation entnommen werden.

2. Verschlüsselung

Da in der Konzeption und Implementierung Daten über das Netzwerk sicher übertragen werden sollen, werden in diesem Kapitel Grundlagen zur Verschlüsselung, sowie zu den verschiedenen Verschlüsselungstechniken erläutert, wobei durch Kombination der Verschlüsselungstechniken ein optimales Maß an Performance und Sicherheit erreicht werden soll.

2.1. Was ist Verschlüsselung?

Daten, die nur für einen bestimmten Empfänger bestimmt sind, müssen in eine für dritte unlesbare Form gebracht werden. Genauer gesagt, werden die Daten einer mathematischen Transformation unterworfen, die es dritten unmöglich machen soll, aus den transformierten (verschlüsselten) Daten auf die Originaldaten zu schließen. Dem legitimen Empfänger soll es jedoch möglich sein, durch Anwendung einer inversen Transformation, wieder die Originaldaten zu erhalten. Für den Einsatz in der Praxis gibt es zahlreiche Implementierungen solcher Transformationen, die sich schon jahrelang bewährt haben, d.h. deren Unsicherheit bis jetzt nicht bewiesen werden konnte. Eingabegrößen dieser Transformationsverfahren sind die zu verschlüsselnden Daten, sowie ein Schlüssel, der durch mathematische Operationen mit den Daten verknüpft wird, wobei durch diese Verknüpfung die verschlüsselten Daten entstehen. Im Gegensatz dazu stehen die Einweg-Hashfunktionen, auf dessen Originaldaten selbst der legitime Empfänger nicht schließen können soll (siehe Punkt 2.3.3). Des Weiteren benötigen schlüsselunabhängige Einweg-Hashfunktionen keinen Schlüssel, sondern nur die Daten als Eingabegröße.
Ein Schlüssel ist in diesem Fall eine sehr große Zahl. Typische Schlüssellängen sind z.B. 128 oder 1024 Bit, wobei jedes Bit die Werte 0 oder 1 annimmt.
In diesem Zusammenhang müssen auch die Begriffe Kryptologie, Kryptographie und Kryptanalyse erläutert werden. Kryptographie bezeichnet die Lehre vom Entwurf von Verschlüsselungsalgorithmen. Die Kryptanalyse könnte als Gegenstück zur Kryptographie bezeichnet werden, da die Kryptanalyse die Kunst beschreibt, eine chiffrierte Nachricht ohne Kenntnis des geheimen

Schlüssels zu entziffern. Die Kryptologie ist der Oberbegriff zu Kryptographie und Kryptanalyse [Wob01].

2.2. Warum ist Verschlüsselung wichtig?

In der heutigen Zeit nehmen die Themen Verschlüsselung und sichere Datenübertragung eine immer wichtiger werdende Rolle ein, da über das Internet immer mehr Dienste angeboten werden, wie z.B. Internet-Banking und Online-Shopping, bei denen persönliche Daten, sowie Daten zum Abschließen von Transfers über das Internet übertragen werden. Es gibt zwar noch viele andere Beispiele bzw. Dienste, wie den Email-Verkehr, bei denen Daten übertragen werden, die nicht für jedermann zugänglich sein sollten, aber im privaten Bereich sind Internet-Banking und Online-Shopping die Dienste, bei denen ein sehr großer Schaden, z.B. durch Geldtransfers, die nicht vom Kontoinhaber getätigt wurden, angerichtet werden kann. Dass Firmen Informationen besitzen, die ebenfalls nicht in falsche Hände geraten dürfen, steht zweifellos außer Frage. Zu diesen Informationen zählen z.B. Strategiepapiere, Umsatzprognosen, Produktdetails, Forschungsdaten und Personaldaten. Diese Daten dürfen nicht für jeden zugänglich sein. Aus diesem Grund müssen sie so geschützt werden, dass nur Berechtigte darauf Zugriff haben bzw. diese Daten verarbeiten können. Ein probates Mittel hierfür ist die Verschlüsselung. Nur wer den geheimen Schlüssel besitzt, kann die Daten lesen. Des Weiteren ist es mit Hilfe von Verschlüsselungstechniken auch möglich, Daten vor Veränderungen zu schützen oder Daten zu beglaubigen. Schließlich ist es nicht erwünscht, dass ein Vertrag, der unterschrieben wurde, im Nachhinein noch geändert wird.
Verschlüsselungstechniken sind also wichtig, um Daten nur bestimmten Personen verfügbar zu machen, die Unveränderbarkeit (Integrität) der Daten zu gewährleisten, sowie eine Beglaubigung der Daten zu ermöglichen, wobei sich eine Person nicht als eine andere ausgeben können soll (Authentifizierung). Somit kann ein Sender später nicht leugnen, dass er eine Nachricht verschickt hat (Verbindlichkeit) [Bur01].

2.3. Verschlüsselungstechniken

Schon zu Zeiten Cäsars wurden verschlüsselte Nachrichten versendet. Die Verschlüsselung bestand allerdings nur darin, die einzelnen Buchstaben der Nachricht durch Buchstaben zu substituieren, die drei Stellen weiter hinten im Alphabet zu finden waren – bekannt auch als Cäsar-Methode. Dass dieser Algorithmus Kryptanalytiker nur für eine kurze Zeit beschäftigt, dürfte jedem klar sein. Anhand der Häufigkeit des Auftretens verschiedener Buchstaben und Buchstabenkombinationen, wird das Geheimnis des Algorithmus schnell gefunden. Die einzige Sicherheit des Algorithmus besteht nur durch dessen Geheimhaltung [Wob01]. Dieser Algorithmus spielt in der heutigen Zeit, zum sicheren Verschlüsseln von Daten, selbstverständlich keine Rolle mehr.

In diesem Abschnitt werden verschiedene Verschlüsselungstechniken vorgestellt, dessen Kryptanalyse noch nicht erfolgreich war, d.h. dessen Unsicherheit bis jetzt nicht bewiesen werde konnte. Nur mit Hilfe des geheimen Schlüssels ist es möglich, auf die unverschlüsselten Daten zu schließen. Des Weiteren wird in diesem Abschnitt beschrieben, wie die verschiedenen Verschlüsselungstechniken kombiniert werden können, um ein optimales Maß an Sicherheit und Performance zu gewährleisten. Die Verschlüsselungsalgorithmen, wie Rijndael, RSA und SHA-1, die hier beschrieben werden, finden in der Konzeption und Implementierung (Kapitel drei und vier) ihre Verwendung und werden aus diesem Grund in diesem Abschnitt erläutert.

Da die Verschlüsselung nicht Hauptthematik dieser Arbeit ist, werden nur die Verschlüsselungstechniken und -algorithmen beschrieben, die am häufigsten in der Praxis eingesetzt werden und als die sichersten gelten. Eine umfassendere Beschreibung der Verschlüsselungsmöglichkeiten würde sonst den Rahmen der Arbeit sprengen.

2.3.1. Symmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung wird zum Ver- und Entschlüsseln von Daten der gleiche Schlüssel benutzt, was in Abbildung 11[Ben02] dargestellt ist. Diese Verschlüsselungsmethode bietet den Vorteil, dass die Ver- und Entschlüsselung schneller als z.B. bei der asymmetrischen Verschlüsselung durchgeführt werden kann. Daher wird versucht, ein Großteil der zu übertragenden Daten mit Hilfe dieser Methode zu verschlüsseln.

Abbildung 11 : graphische Darstellung des Prinzips der symmetrischen Verschlüsselung

Ein Nachteil dieser Methode ist das Problem des Schlüsselaustausches. Erzeugt z.B. A einen Schlüssel, so muss B dieser Schlüssel mitgeteilt werden. Dies sollte verschlüsselt geschehen. An dieser Stelle setzt die asymmetrische Verschlüsselung ein [Ben02].

2.3.1.1. AES (Rijndael)

1997 wurde ein Wettbewerb ausgeschrieben, in dem eine Nachricht, die durch den damaligen symmetrischen Standard-Verschlüsselungsalgorithmus DES[1] mit einem 48-Bit-Schlüssel verschlüsselt wurde, durch Kompromittierung dieses Schlüssels entschlüsselt werden sollte. Der Sieger erreichte das Ziel in einer Zeit von 280 Stunden. Spätestens zu diesem Zeitpunkt war es Wissenschaftlern, Industriellen und Behörden klar, dass DES als Standard-Verschlüsselungsalgorithmus abgelöst werden musste [Bur01]. Aus diesem Grund führte

[1] Data Encryption Standard – wurde 1976 zum offiziellen Verschlüsselungsstandard erklärt und arbeitet mit einer maximalen Schlüssellänge von 56 Bit

das NIST[1] im April 1997 einen offenen Workshop durch, in dem die Anforderungen des neuen offiziellen Verschlüsselungsalgorithmus AES (Advanced Encryption Standard) diskutiert wurden. Der Algorithmus sollte folgende Kriterien erfüllen [Wob01]:

- Es muss ein symmetrischer Blockalgorithmus sein.
- Der Algorithmus muss eine Schlüssellänge von 128, 192 und 256 Bit unterstützen und eine Blocklänge von 128 Bit verwenden.
- Er soll leicht in Hardware und Software implementierbar sein und eine gute Performance bieten.
- AES soll unempfindlich gegen allen bekannten Methoden der Kryptanalyse sein.
- Durch einen Einsatz auf Smartcards soll der Algorithmus geringe Ressourcen beanspruchen.
- Damit jeder diesen Algorithmus nutzen darf, muss er frei von patentrechtlichen Ansprüchen sein.

Am 15.4.99 endeten die öffentlichen Begutachtungen aller Kandidaten. Fünf Algorithmen (MARS, RC6, Rijndael, Serpent, Twofish) kamen in die engere Wahl. Die fähigsten Kryptanalytiker wurden nun damit beschäftigt, diese Algorithmen zu begutachten. Am 2.10.00 gab das NIST den neuen AES bekannt – Rijndael. Ein Algorithmus, der von den belgischen Autoren Joan Daemen und Vincent Rijmen entwickelt wurde [Wob01].

Der Algorithmus

Da der Algorithmus sehr komplex ist, wird in diesem Abschnitt nur die grundlegende Arbeitsweise von Rijndael beschrieben. Eine komplette detaillierte Beschreibung von Rijndael ist unter [Dae99] oder [Hun01] zu finden. In [Hun01] ist sogar jeder Schritt des Algorithmus mit einem Java-Applet

[1] National Institute of Standards and Technology – eine Abteilung des US-Handelsministeriums, die u.a. kryptographische Standards festlegt

graphisch hinterlegt, so dass die einzelnen Schritte interaktiv nachvollzogen werden können.

Im Folgenden wird der Rijndael-Algorithmus für 128-Bit-Blöcke und der Verwendung eines 128-Bit-Schlüssels beschrieben. Abbildung 12[1][Wob01] stellt den Ablauf der Verschlüsselung graphisch dar.

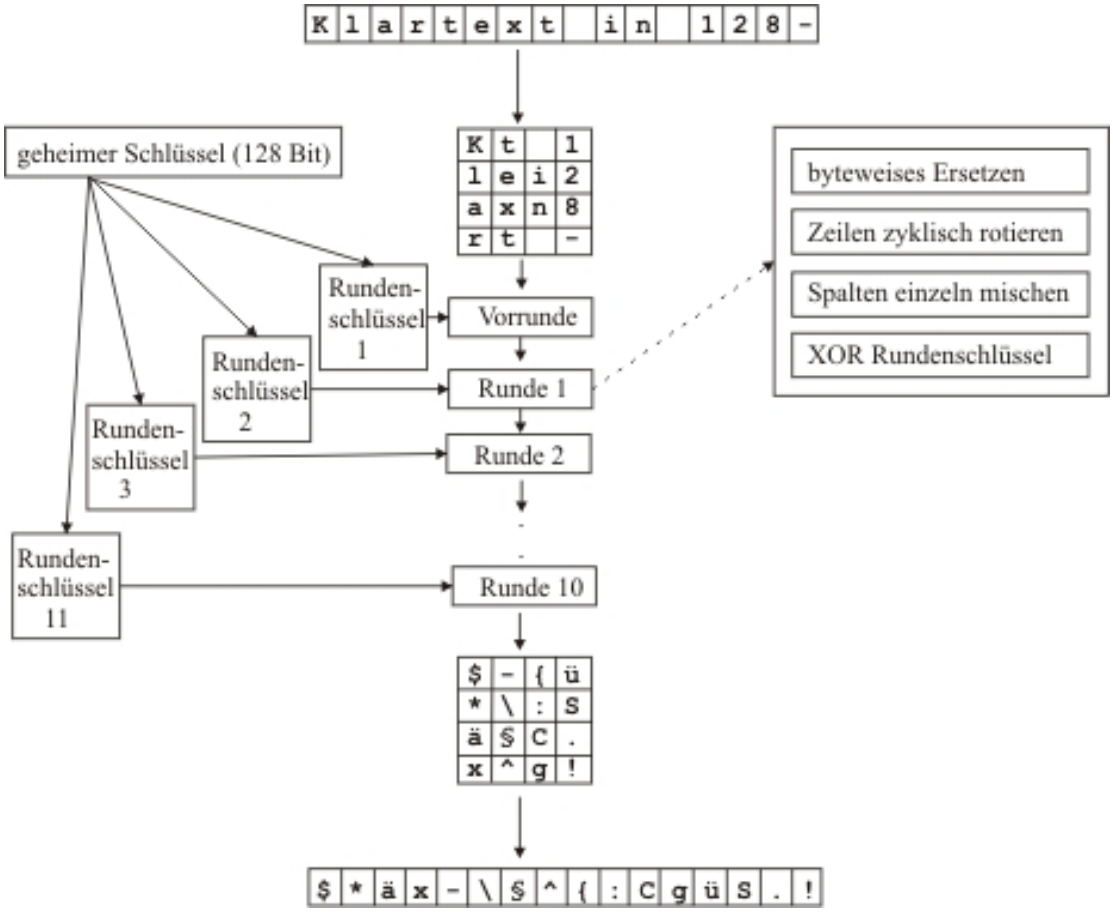


Abbildung 12 : graphische Darstellung des Ablaufes der Verschlüsselung mit Hilfe von Rijndael

Zuerst wird der zu verschlüsselnde Block, der aus 128-Bit bzw. 16 Byte besteht, spaltenweise in eine 4x4-Matrix (State-Matrix) geschrieben. Vor der ersten Runde steht der Klartextblock in der Matrix. Am Ende der zehnten Runde befindet sich der verschlüsselte Block in der Matrix, der dann spaltenweise ausgelesen wird. Vor der Verschlüsselung werden aus dem 128-Bit-Schlüssel

[1] Diese Abbildung entspricht nicht genau der originalen, da diese Fehler aufweist. Vor den zehn Runden gibt es noch eine Vorrunde, die den ersten Rundenschlüssel benutzt. Damit gibt es elf anstatt, wie fälschlicher Weise in der originalen Abbildung dargestellt, zehn Rundenschlüssel.

elf Rundenschlüssel erzeugt. Dies wird Schlüsselexpansion genannt, dessen Algorithmus in [Dae99] nachgelesen werden kann.

In jeder Runde des Rijndael-Algorithmus werden folgende Schritte durchgeführt [Wob01][Hun01][Dae99]:

- *Substitution*: Die einzelnen Bytes der State-Matrix werden nach einem festen bekannten Schema substituiert.
- *ShiftRow*: Die einzelnen Zeilen der Matrix werden byteweise zyklisch nach links, um 0(1.Zeile), 1, 2 oder 3(4.Zeile) Bytes, rotiert.
- *MixColumn*: Die Bits werden spaltenweise durchmischt, indem die Einträge der einzelnen Spalten mit Konstanten multipliziert werden.
- *AddRoundKey*: Der je nach Runde bereitgestellte Schlüssel wird mit den einzelnen Einträgen in der State-Matrix bitweise XOR verknüpft.

In der Vorrunde wird als einzige Operation ein *AddRoundKey* mit dem ersten Rundenschlüssel durchgeführt. In der letzen (zehnten) Runde wird kein *MixColumn* durchgeführt. Diese Runde wird auch Schlussrunde genannt.
Rijndael kann auch mit 192- und 256-Bit-Blöcken arbeiten. Dabei ändert sich die Größe der State-Matrix auf 4x6 bzw. 4x8. Die Anzahl der Stellen, die bei der Linksrotierung verschoben werden, kann aus der Tabelle 17 entnommen werden [Hun01][Dae99].

Zeile	**Anzahl der Linksverschiebungen**		
	4x4-Matrix	*4x6-Matrix*	*4x8-Matrix*
1	0	0	0
2	3	3	4
3	2	2	3
4	1	1	1

Tabelle 17 : Anzahl der Linksverschiebungen bei ShiftRow

Je nach Verwendung der Schlüssel- und Blocklängen, verändert sich natürlich auch die Anzahl der Runden, welche in Tabelle 18 dargestellt ist. Die Anzahl der Runden ist so festgelegt, dass eine optimale Durchmischung der Bits erreicht wird.

Schlüssellänge	Anzahl der Runden		
	128-Bit Blocklänge	*192-Bit Blocklänge*	*256-Bit Blocklänge*
128 Bit	10	12	14
192 Bit	12	12	14
256 Bit	14	14	14

Tabelle 18 : Anzahl der Verschlüsselungsrunden in Abhängigkeit von Schlüssel- und Blocklänge

Beim Entschlüsseln eines Blocks werden die Operationen in umgekehrter Reihenfolge vollzogen, d.h., die Schlussrunde ist die erste Runde und *AddRoundKey* die erste Operation. Beim *ShiftRow* wird demzufolge in die entgegengesetzte Richtung rotiert [Hun01][Dae99].

Sicherheit von Rijndael

Zurzeit sind keine wirksamen Angriffe gegen Rijndael bekannt. Die einzige Möglichkeit, an den geheimen Schlüssel und somit an den entschlüsselten Text zu gelangen besteht darin, alle möglichen Schlüssel zu erzeugen. Dieser Versuch, an den Schlüssel zu gelangen wird Brute-Force-Angriff genannt. Wie schon am Anfang dieses Abschnittes beschrieben wurde, gelang es 1997 DES unter Verwendung eines 48-Bit-Schlüssels in 280 Stunden zu brechen. In einem weiteren Wettbewerb wurde DES unter Verwendung eines 56-Bit-Schlüssels (höchstmöglicher Schlüssel bei DES) 1999 in nur 24 Stunden gebrochen. Dabei kam ein so genannter DES-Cracker[1] zum Einsatz.
Durchschnittlich muss 50 Prozent des Schlüsselraums durchsucht werden, bis der richtige gefunden wurde. Im einem Worst-Case-Szenario, das von dem NIST aufgestellt wurde, wird angenommen, dass für die Untersuchung von einem Prozent des Schlüsselraumes bei der Verwendung von 56-Bit-Schlüsseln eine Sekunde benötigt wird und bei 50 Prozent eine Minute. Jedes Mal, wenn der Schlüssel um ein Bit verlängert wird, verdoppelt sich dessen Bereich und somit auch die Zeit zum Finden eines solchen Schlüssels, d.h. bei einem 57-Bit-Schlüssel dauert es demzufolge doppelt so lange diesen Schlüssel zu finden,

[1] Dabei handelt es sich um einen speziellen Computer, der nur zu einem Zweck erschaffen wurde: zum Testen von DES-Schlüsseln.

wie bei einer Verwendung eines 56-Bit-Schlüssel; in diesem Szenario also zwei Minuten. In Tabelle 19 sind die verschiedenen Zeiten des Worst-Case-Szenarios zum Finden eines Schlüssels bei einem Brute-Force-Angriff unter Verwendung verschiedener Schlüssellängen dargestellt.

Schlüsselbits	1 % des Schlüsselraums	50 % des Schlüsselraums
56	1 Sekunde	1 Minute
57	2 Sekunden	2 Minuten
64	4,2 Minuten	4,2 Stunden
72	17,9 Stunden	44,8 Stunden
80	190,9 Tage	31,4 Jahre
90	535 Jahre	321 Jahrhunderte
108	140.000 Jahrtausende	8 Millionen Jahrtausende
128	146 Milliarden Jahrtausende	8 Billionen Jahrtausende

Tabelle 19 : Zeit, die zum Finden eines Schlüssels bei einem Brute-Force-Angriff unter Verwendung verschiedener Schlüssellängen benötigt wird

Obwohl sich die Technik sehr schnell weiterentwickelt, ist es kaum denkbar, dass sie so weit fortschreitet, dass es in naher Zukunft möglich ist, Rijndael in einer realistischen Zeit zu brechen, wobei es zu bedenken gilt, dass Rijndael eine Schlüssellänge bis 256 Bit unterstützt.

2.3.2. Asymmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung werden für die Ver- und Entschlüsselung verschiedene Schlüssel benutzt. Diese beiden Schlüssel werden als Private Key (privater Schlüssel) und Public Key (öffentlicher Schlüssel) bezeichnet. Daten, die mit dem Public Key verschlüsselt werden, können nur mit dem Private Key entschlüsselt werden und umgekehrt, was in Abbildung 13[Ben02] und Abbildung 14[Ben02] dargestellt ist. Der Public Key ist der Schlüssel, der jedem zugänglich gemacht werden kann, da mit ihm verschlüsselte Daten nur mit Hilfe des Private Key entschlüsselt werden können. Demzufolge ist der Private Key geheim zu halten [Ben02].

Abbildung 13 : graphische Darstellung des Prinzips der asymmetrischen Verschlüsselung durch den Besitzer des Private Key

Abbildung 14 : graphische Darstellung des Prinzips der asymmetrischen Verschlüsselung durch den Besitzer des Public Key

Ein Nachteil dieser Methode besteht im hohen Rechenaufwand beim Ver- und Entschlüsseln von Daten. Deshalb wird diese Methode benutzt, um am Anfang einer Kommunikation den Schlüssel für die symmetrische Verschlüsselung auszutauschen, mit dem dann die zu sendenden Daten verschlüsselt werden.

Der höhere Rechenaufwand beruht darauf, dass asymmetrische Verschlüsselungsverfahren meist auf Primzahlen beruhen. Da diese nicht so dicht beieinander liegen wie natürliche Zahlen, müssen größere Schlüssel bei der Verwendung der asymmetrischen Verschlüsselung gewählt werden, um die gleiche Sicherheit gewährleisten zu können, wie bei der symmetrischen Verschlüsselung. Durch die Verwendung größerer Schlüssel sind die mathematischen Operationen aufwendiger, kosten also mehr Zeit. Während Schlüssellängen von 128 Bit bei der symmetrischen Verschlüsselung als sicher gelten, sind typische Schlüssellängen bei asymmetrischen Verfahren 1024 oder 2048 Bit. Die Kombination dieser beiden Verschlüsselungstechniken, bei der die Vorteile beider Verfahren genutzt und die Nachteile beseitigt werden, wird als hybride Verschlüsselung bezeichnet [Ben02][Bra00].

2.3.2.1. RSA

Eine der bekanntesten und am meisten verwendete asymmetrische Verschlüsselungsmethode ist RSA. Diese Methode wurde 1978 von Ronald Rivest, Adi Shamir und Leonard Adleman veröffentlicht.
RSA nutzt die Tatsache, dass es schwierig bzw. sehr zeitaufwendig ist, große Zahlen in ihre Primfaktoren zu zerlegen. So besteht bei diesem Verfahren der Public Key im Wesentlichen aus dem Produkt zweier großer Primzahlen. Um nun vom Public Key auf den Private Key schließen zu können, müsste eine solche Zerlegung gefunden werden. Dies ist sehr aufwendig, da alle heute bekannten Methoden zur Faktorisierung einer Zahl in Abhängigkeit von der Größe dieser Zahl exponentielle Zeit benötigen [Sch96].

Der Algorithmus

Um sich ein Schlüsselpaar zu generieren werden als erstes 2 große Primzahlen p und q erzeugt. Die Schlüssellänge der zu erzeugenden Schlüssel berechnet sich aus der Summe der Längen der beiden Primzahlen, wobei RSA eine Schlüssellänge bis 2048 Bit unterstützt. Das Produkt dieser beiden Primzahlen

wird als *n* bezeichnet. Danach wird ein zufälliger Chiffrierschlüssel *e* gewählt, der relativ prim zum Produkt *(p-1)(q-1)* ist, d.h. der Chiffrierschlüssel *e* darf keine gemeinsamen Faktoren mit diesem Produkt besitzen. Nun kann der Dechiffrierschlüssel *d* mit der Formel

$$d = e^{-1} \bmod((p-1)(q-1))$$

berechnet werden. Der Kehrwert einer Modulo-Operation, wie zur Berechnung von *d* notwendig, wird mit Hilfe des erweiterten Euklidischen Algorithmus realisiert. Um eine eindeutige Lösung zu finden, ist es notwendig, dass *e* und das Produkt *(p-1)(q-1)* relativ prim zueinander sind. Besitzen *e* und das Produkt *(p-1)(q-1)* einen gemeinsamen Faktor, so gibt es für dieses Problem keine Lösung. Einzelheiten über den erweiterten Euklidischen Algorithmus können in [Knu81] nachgelesen werden [Wob01][Sch96].
Mit Hilfe der Formel

$$c = (m^e) \bmod(n)$$

wird der zu verschlüsselnde Block *m* zum Block *c* verschlüsselt. Eine Entschlüsselung wird durch die Formel

$$m = (c^d) \bmod(n)$$

erreicht. Der Block könnte genauso gut auch mit *d* chiffriert und mit *e* dechiffriert werden.
Sender und Empfänger einer Nachricht müssen demnach den Wert *n* kennen, wobei *e* nur dem Empfänger bekannt sein muss und *d* nur dem Sender bekannt sein darf. Demnach werden *e* und *n* als Public und *d* und *n* als Private Key definiert [Wob01].

Sicherheit von RSA

Wie schon am Anfang dieses Abschnittes erwähnt wurde, besteht die Sicherheit von RSA darin, dass große Zahlen in einer realistischen Zeit nicht faktorisiert werden können. Während sich bei Brute-Force-Angriffen die Zeit zum Finden eines Schlüssels bei Hinzunahme eines Bits verdoppelt, vergrößert sich die Zeit, die zum Faktorisieren dieser Zahl benötigt wird, um den Faktor 1035 bis 1036. Demzufolge ist der Brute-Force-Angriff der effizienteste Angriff [Bur01].
Im August 1999 wurde ein 512-Bit-Schlüssel gebrochen. Dazu waren fünf Monate und 292 handelsübliche Standard-Computer notwendig [RSA99]. Im April 2000 veröffentlichten die RSA Labs einen Bericht, indem sie eine Übersicht aufstellten, wie lange es dauern würde, verschiedene Schlüssellängen zu brechen. In dieser Übersicht wird ebenfalls die Sicherheit von RSA und symmetrischer Verschlüsselungsmethoden gegenübergestellt, indem z.B., durch Forschungen belegt, symmetrische Schlüssel mit 128 Bit so sicher wie RSA-Schlüssel mit 1620 Bit betrachtet werden. In Tabelle 20 ist diese Übersicht auszugsweise dargestellt [Sil01][Bur01].

Symmetrischer Schlüssel in Bit	RSA-Schlüssel in Bit	Zeit zum Brechen des Schlüssels
56	430	< 5 Minuten
80	760	600 Monate
96	1020	3 Millionen Jahre
128	1620	10^{16} Jahre

Tabelle 20 : Zeitaufwand zum Brechen von Schlüsseln verschiedener Längen

2.3.3. Digitale Unterschriften

Um zu überprüfen, ob eine empfangene Nachricht wirklich vom angegebenen Absender stammt und auch nicht während der Übertragung manipuliert wurde, kann unter Einsatz der asymmetrischen Verschlüsselung über eine so genannte Einweg-Hashfunktion eine digitale Unterschrift vom Absender erzeugt werden.
Eine Hashfunktion ist eine mathematisch oder anderweitig definierte Funktion, die einen Eingabestring variabler Länge in einen kürzeren Ausgabestring fester Länge (den Hashwert) umwandelt. Die Hashfunktion soll dabei zwei Eigenschaften besitzen. Zum einen darf es nicht möglich sein, vom Ausgabestring auf den Eingabestring zu schließen und zum anderen sollen ähnliche Eingabestrings verschiedenste Ausgabestrings zur Folge haben.
Nach der Erstellung des Hashwertes einer Nachricht wird dieser mit dem privaten Schlüssel verschlüsselt und zusammen mit der verschlüsselten Nachricht versendet. Der Empfänger erstellt von der entschlüsselten Nachricht einen Hashwert und vergleicht diesen mit dem vom Sender gesendeten Hashwert, den der Empfänger mit Hilfe des öffentlichen Schlüssels entschlüsselt. Sind beide Werte identisch, wurde die gesendete Nachricht nicht manipuliert.
Der verschlüsselte Hashwert wird hierbei als digitale Unterschrift oder digitale Signatur bezeichnet. Der Hashwert könnte auch mit Hilfe einer symmetrischen Methode verschlüsselt werden. Allerdings setzt dies voraus, dass der Empfänger den symmetrischen Schlüssel kennen muss, der ja bei Empfänger und Sender gleich ist. Da aufgrund der schnelleren Ver- und Entschlüsselung von Daten symmetrische Verschlüsselungen verwendet werden, müsste entweder der Sitzungsschlüssel oder ein neuer Schlüssel zum Ver- und Entschlüsseln des Hashwertes verwendet werden. Der asymmetrische Schlüssel ist bereits bekannt, da er zum Austausch des Sitzungsschlüssels verwendet wird. Des Weiteren bietet eine Signierung mit einer asymmetrischen Verschlüsselung eine höhere Sicherheit, da nur derjenige die Unterschrift ändern kann, der in Besitz des Private Key ist und den weiß nur ein Kommunikationspartner, wogegen der symmetrische Schlüssel zum Ver- und Entschlüsseln beiden bekannt ist. [Ben02][Sch96][Wob01].

Die Erstellung einer digitalen Unterschrift ist in Abbildung 15[Ben02] und die Überprüfung der digitalen Unterschrift in Abbildung 16[Ben02] dargestellt.

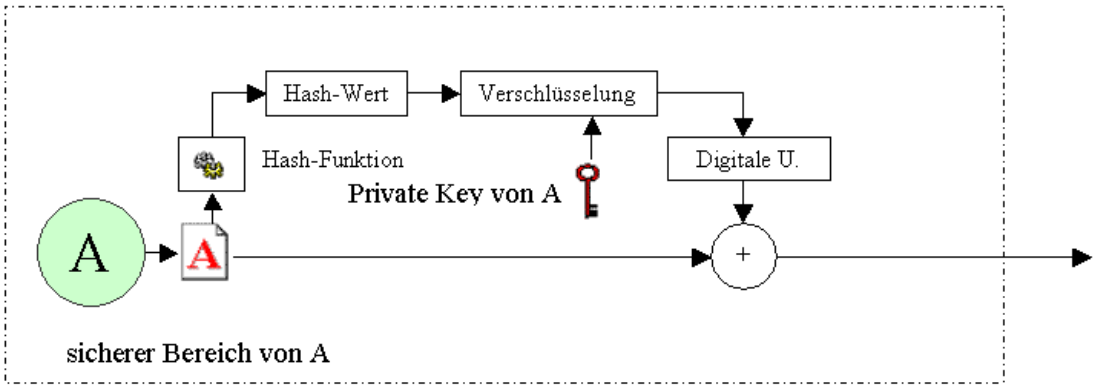


Abbildung 15 : graphische Darstellung des Ablaufs der Erstellung einer digitalen Unterschrift

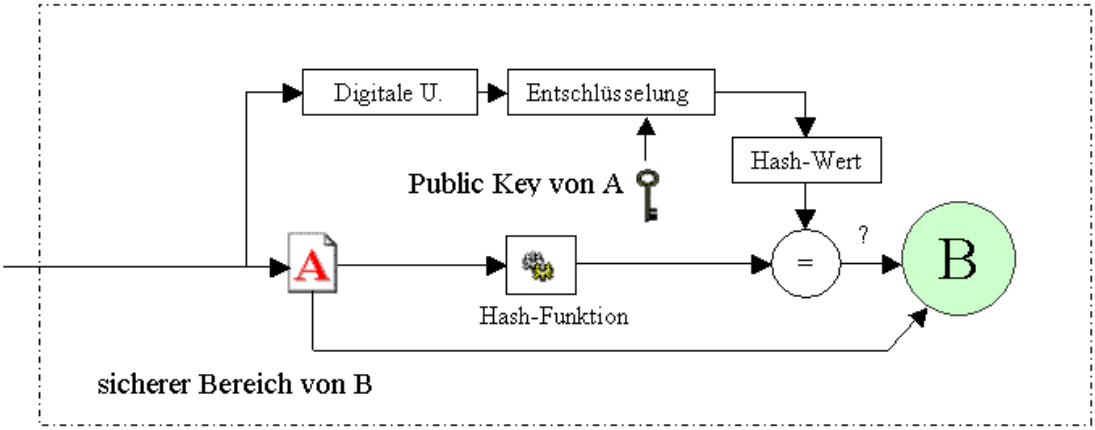


Abbildung 16 : graphische Darstellung des Ablaufs der Überprüfung einer digitalen Unterschrift

2.3.3.1. SHA-1

SHA-1 (*Secure Hash Algorithm Version 1*) wurde von dem NIST und der NSA[1] entwickelt und 1994 publiziert. SHA-1 ist eine Einweg-Hashfunktion, die einen 160-Bit-langen Hashwert liefert. Dieser Algorithmus gilt als einer der wenigen Algorithmen, gegen die keine Angriffe bekannt sind. Deshalb findet SHA-1 immer mehr Anwendung und wird in bekannten Protokollen, wie SSL oder PGP eingesetzt, indem es die bis dato eingesetzte Einweg-Hashfunktion MD5 ersetzt, in der unter anderem Sicherheitslücken entdeckt wurden [Wob01].

Der Algorithmus

Zuerst wird die Nachricht, von der der Hashwert erzeugt werden soll, so aufgefüllt, dass die Länge einem Vielfachen von 512 Bit minus 64 Bit entspricht. Das Auffüllen läuft so ab, dass zuerst eine eins und dann so viele Nullen angehangen werden, dass beim letzten 512-Bit-Nachrichtenblock noch 64 Bit übrig bleiben. In diese 64 Bit wird die Länge der Nachricht gespeichert.
Bei SHA-1 werden fünf Variablen der Länge 32 Bit initialisiert:

A=0x67452301
B=0xefcdab89
C=0x98babcfe
D=0x10325476
E=0xc3d2e1f0

Am Ende des Algorithmus steht der Hashwert in diesen Variablen, welcher eine Länge von fünf mal 32 Bit, also 160 Bit, besitzt.
Die Hauptschleife des Algorithmus verarbeitet jeweils 512 Bit der Nachricht und wird für jeden Nachrichtenblock durchlaufen. Sie besteht für jeden Nachrichtenblock aus vier Runden mit jeweils 20 Operationen, also insgesamt 80 Operatio-

[1] National Security Agency: geheime amerikanische Behörde, die sich intensiv mit Kryptologie, sowie mit der weltweiten Überwachung und nachrichtendienstlichen Gewinnung von Daten beschäftigt [Wob01].

nen. Für jeden Nachrichtenblock kann die Hauptschleife in folgendem Pseudocode ausgedrückt werden [Sch96]:

```
a=A, b=B, c=C, d=D, e=E
Für t=0 bis 79
        Temp=(a <<< 5) + f_t(b,c,d) + e + W_t + K_t
        e=d
        d=c
        c=b <<< 30
        b=a
        a=Temp
A=A+a, B=B+b,C=C+c,D=D+d,E=E+e
```

In dieser Hauptschleife werden zuerst die Variablen *A, B, C, D* und *E* in die Hilfsvariablen *a, b, c, d* und *e* kopiert. Die darauf folgende Schleife wird für jeden Nachrichtenblock 80-mal durchlaufen. Der Inhalt der Hilfsvariablen *Temp* wird durch additive Verknüpfung der permutierten Hilfsvariable *a*, dem Funktionswert der nichtlinearen Funktion f_t, der Hilfsvariablen *e*, dem expandierten Nachrichtenblock W_t und der Variablen K_t gebildet. Danach werden die Hilfsvariablen, wie im Pseudocode dargestellt, initialisiert. Nach Beendigung dieser Schleife werden die Variablen *A, B, C, D* und *E* durch additive Verknüpfung mit den Variablen *a, b, c, d* und *e* verändert.
Die Berechnungsvorschriften der in der Hauptschleife verwendeten Variablen und Funktionen werden in Abhängigkeit der Schleifendurchlaufzahl im nun folgenden Abschnitt erläutert.
t ist die Operationsnummer, die für die vier Runden mit je 20 Operationen (insgesamt 80 Operationen) durchlaufen wird. Die Operation <<< *s* beschreibt eine zirkuläre Linksverschiebung um *s* Bit. K_t ist eine Variable, die wie folgt initialisiert wird:

K_t = 0x5a827999 (entspricht $2^{32} * 2^{1/2} / 4$) für t = 0 bis 19

K_t = 0x6ed9eba1 (entspricht $2^{32} * 3^{1/2} / 4$) für t = 20 bis 39

K_t = 0x8f1bbcdc (entspricht $2^{32} * 5^{1/2} / 4$) für t = 40 bis 59

K_t = 0xca62c1d6 (entspricht $2^{32} * 10^{1/2} / 4$) für t = 60 bis 79

W_t steht für das *t*-te 32-Bit-Wort des aktuellen Nachrichtenblockes, der mit Hilfe des folgenden Algorithmus von 16 32-Bit-Wörtern (M_0 bis M_{15} = der 512-Bit-Nachrichtenblock) auf 80 32-Bit-Wörter (W_0 bis W_{79}) expandiert wird:

$W_t = M_t$ für t = 0 bis 15

$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1$ für t = 16 bis 79

Die nichtlinearen Funktionen f_t lauten wie folgt:

$f_t(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$ für t = 0 bis 19

$f_t(X,Y,Z) = X \oplus Y \oplus Z$ für t = 20 bis 39

$f_t(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$ für t = 40 bis 59

$f_t(X,Y,Z) = X \oplus Y \oplus Z$ für t = 60 bis 79

Sicherheit von SHA-1

Die zwei wichtigsten Kriterien für die Sicherheit von SHA-1 sind zum einen, dass keine kryptographischen Angriffe gegen SHA-1 bekannt sind und zum anderen, dass SHA-1 einen 160-Bit-langen Hashwert liefert. Somit bietet SHA-1 einen besseren Schutz gegen einen Brute-Force-Angriff, als z.B. MD5, der nur 128-Bit-lange Hashwerte liefert. Des Weiteren bietet SHA-1 eine bessere Performance bei der Erzeugung von Hashwerten gegenüber anderen Einweg-Hashfunktionen, die ebenfalls 160-Bit-lange Hashwerte liefern. Ein weiteres Kriterium, das für SHA-1 spricht ist, dass Sicherheitslücken in MD5 entdeckt wurden und MD5 somit nicht mehr als sicher gilt [Wob01].

2.3.4. Zertifikate

Um überprüfen zu können, ob ein von der Gegenseite empfangener öffentlicher Schlüssel wirklich der öffentliche Schlüssel des Senders ist, werden Zertifikate eingesetzt. Zertifikate werden von Zertifizierungsstellen ausgestellt und sind mit einem Ausweis vergleichbar. Ein Zertifikat enthält im Wesentlichen den Namen der Stelle, deren Authentizität durch diesen Ausweis bestätigt wird, deren Public Key, den Namen der ausstellenden Zertifizierungsstelle und deren digitale Unterschrift über das Zertifikat. Ein schematischer Überblick über ein standardisiertes Zertifikat ist in Abbildung 17[Ben02] dargestellt. Die einzige Aufgabe eines Zertifikates ist die eindeutige Zuordnung eines Public Key zu einer bestimmten Institution. Da das Zertifikat mit einer digitalen Unterschrift versehen ist, ist es ohne Kenntnis des privaten Schlüssels der Zertifizierungsstelle unmöglich das Zertifikat zu verändern.

Da ein Zertifikat von einer Zertifizierungsstelle ausgestellt wurde, muss dieser Stelle vertraut werden. Um ein Zertifikat von einer geeigneten Zertifizierungsstelle zu erhalten, ist es notwendig, sich bei dieser Stelle auf konventionelle Art und Weise, z.B. mit Hilfe eines Lichtbildausweises, auszuweisen. Somit ist sichergestellt, dass ein Zertifikat auch wirklich der Institution zugeordnet wurde, die sich mit den in dem Zertifikat angegebenen Daten ausgewiesen hat. Damit der Aufwand bzw. der Anreiseweg für den Antragsteller eines Zertifikats nicht zu groß bzw. zu lang wird, ist eine große Anzahl von Zertifizierungsstellen erwünscht. Um nicht den Überblick zu verlieren und um zu verhindern, dass somit einer zweifelhaften Stelle vertraut wird, wurde eine Zertifizierungshierarchie eingeführt, welche in Abbildung 18[Ben02] dargestellt ist. Bleibt natürlich die Frage offen, wie das Zertifikat des Root CA überprüft werden kann. Schließlich gibt es keine höhere Zertifizierungsstelle, über die das Zertifikat verifiziert werden könnte. Aus diesem Grund wurden die Public Keys der wichtigsten Zertifizierungsstellen in den Quelltext der Browser integriert. Es muss also den Entwicklern diverser Browser vertraut werden [Ben02].

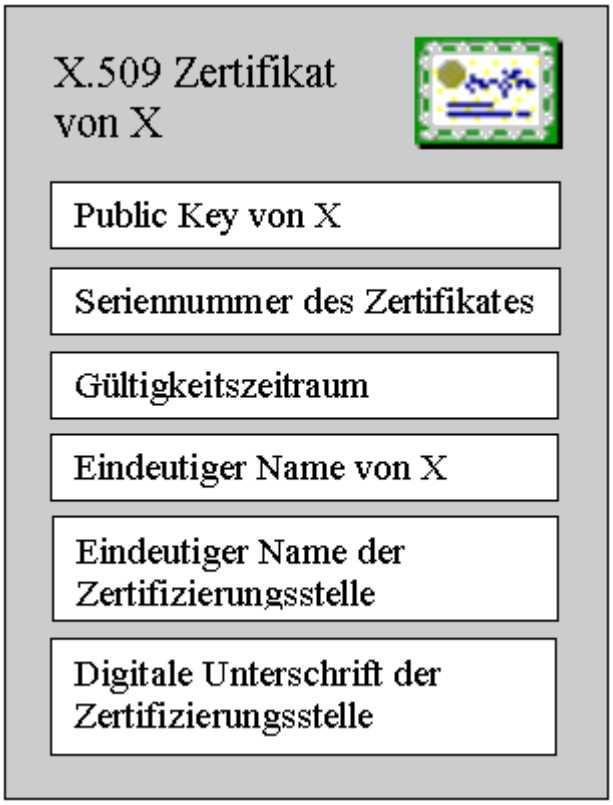


Abbildung 17 : schematische Darstellung eines Zertifikats

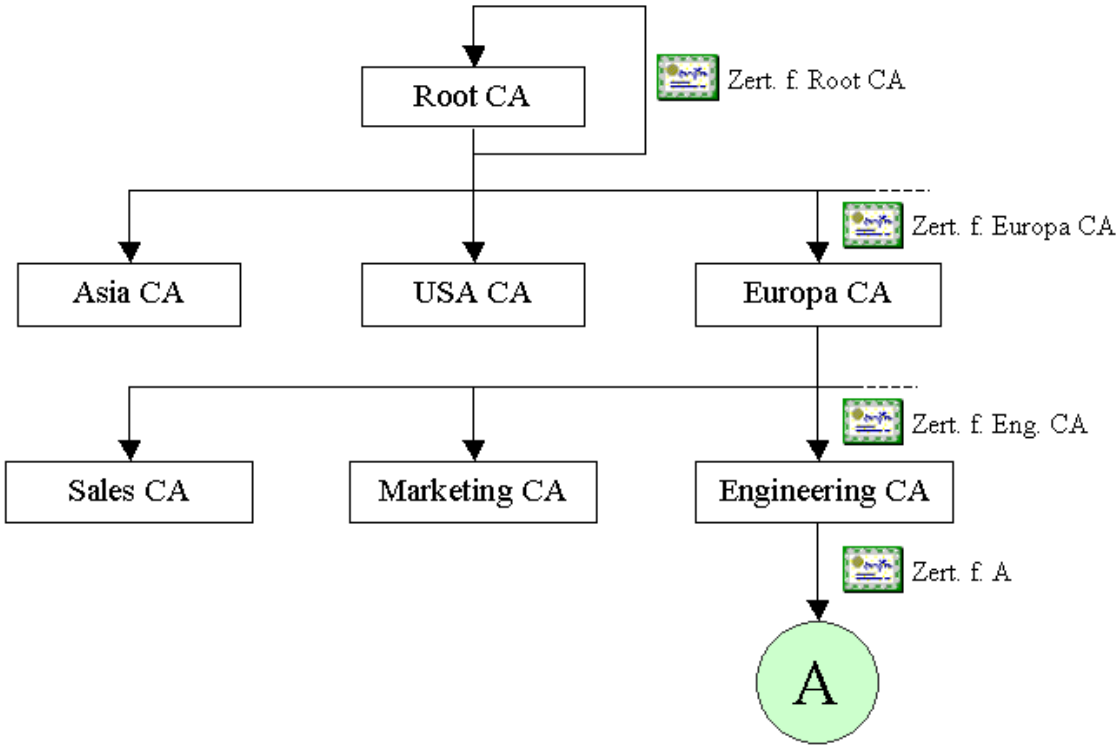


Abbildung 18 : graphische Darstellung der Zertifizierungshierarchie der Zertifizierungsstellen (CA)

3. Konzeption des Systems

3.1. Aufgabe des Systems

Das System bietet die Möglichkeit, Hardware- und Software-Informationen mehrerer Rechner zentral zu erfassen. So kann auf einfache Art- und Weise ein Gesamtüberblick über die Ressourcen der sich im Netzwerk befindlichen Rechner erstellt werden. Die Kommunikationsdaten werden dabei verschlüsselt, sodass Informationen, falls sie über einen unsicheren Weg (z.B. das Internet) ausgetauscht werden, vor ungewolltem Lesen und unbemerktem Ändern geschützt sind.

3.2. Aufbau

3.2.1. Das Serverprogramm

Für den Datenaustausch wird eine Client-Server-Architektur verwendet. Das Serverprogramm läuft dabei auf den Rechnern, dessen Informationen ausgelesen werden sollen. Es ist auf Windows 95/98/ME und Windows 2000/XP lauffähig. Damit das Serverprogramm wenig Speicher beansprucht, werden benötigte DLLs bzw. Klassen erst bei einem Kommunikationsaufbau dynamisch zur Laufzeit geladen bzw. erzeugt. Das Serverprogramm läuft für den Anwender transparent im Hintergrund. Eine Interaktion mit dem Serverprogramm (Anzeigen des Hauptfensters, Beenden des Serverprogramms) ist nur mit Hilfe eines Passworts möglich. Ebenso wird der Zugriff auf Konfigurationseinstellungen des Serverprogramms durch Verschlüsselung dieser Daten verhindert. Des Weiteren ist es nur mit Hilfe des Clientprogramms möglich, Konfigurationsdateien für das Serverprogramm zu erstellen, wodurch Änderungen an der Konfiguration des Serverprogramms, auch bei Kenntnis des Zugriffspasswortes auf das Serverprogramm, nicht vollzogen werden können. Unter Windows 2000 und XP ist es möglich, das Serverprogramm als Dienst in das System zu integrieren, sodass eine Zugriffsbeschränkung auf eine Interaktion mit dem Server, vom Betriebssystem vorgenommen wird.

Eventuelle Ausgaben von Fehlermeldungen werden dabei in das Fehlerprotokoll von Windows geschrieben. Die Version des Serverprogramms, die nicht als Dienst läuft, gibt Fehlermeldungen des Serverprogramms im Hauptfenster der Anwendung aus, welche die beschriebenen Zugriffsbeschränkungen bietet. Um eine Interaktion mit dem Anwender zu ermöglichen, platziert das Serverprogramm ein Icon in der Tray-Leiste von Windows.

3.2.2. Das Clientprogramm

Das Clientprogramm kontaktiert die einzelnen Serverprogramme, empfängt die Ressourcen-Informationen und stellt diese dar. Wie beim Serverprogramm, werden auch die Konfigurationsdaten des Clientprogramms, die für das Ressourcen-Abfrage-System verwendet werden, verschlüsselt gespeichert. Alle Konfigurationsdateien des Ressourcen-Abfrage-Systems werden mit Hilfe des Clientprogramms erstellt. Somit können Änderungen an den Konfigurationen nur durch das Clientprogramm erfolgen. Des Weiteren ist es möglich, Ressourcen-Informationen mehrerer Rechner in eine Datei zu speichern oder aus einer Datei zu laden.
Das Hauptfenster des Clientprogramms zeigt auf der linken Seite in einem Baum die abgefragten Serverprogramme und die Bezeichnungen der abgefragten Ressourcen-Informationen an. Auf der rechten Seite werden in einer tabellenartigen Darstellung die Daten, der auf der linken Seite markierten Ressourcen-Information, angezeigt. Im unteren rechten Bereich werden Informationen zum Status bestimmter Operationen, wie z.B. die Abfrage der Serverprogramme oder Fehlermeldungen angezeigt. Die Auswahl der Aktionen, wie das Anlegen von Konfigurationsdateien oder das Abfragen der Serverprogramme, erfolgt über eine Menüsteuerung.

3.2.3. Ressourcen-Info-DLL

Die Methoden zum Auslesen der Ressourcen-Informationen werden in einer DLL gekapselt. Somit können diese Methoden auch in anderen Programmiersprachen genutzt werden, die das Einbinden von DLLs unterstützen. Damit es keine Probleme mit den Datentypen, die beim Exportieren und Importieren von Daten benutzt werden, gibt, werden nur Datentypen der Windows-API verwendet. Die Methoden der DLL, die exportiert werden, sind für den Nutzer transparent verwendbar, unabhängig von der Windowsversion. Trotz der Komplexität der internen Methoden der DLL, ist der Aufruf der Methoden, die exportiert werden, also die, die vom Nutzer verwendet werden können, so gehalten, dass keine Kenntnisse der internen Strukturen oder Methoden notwendig sind. Als Rückgabe der Ressourcen-Informations-Methoden erhält der Nutzer einen Zeiger auf einen Datentyp, aus denen er problemlos die Ressourcen-Informationen auslesen kann.
Des Weiteren exportiert die DLL Methoden, die eine Ressourcen-Information als String zurückliefern, so dass diese Informationen einfach in eine Datei gespeichert oder über eine Socket-Verbindung versendet werden können. Ebenso exportiert die DLL Methoden, die es ermöglichen diesen String in eine Ressourcen-Information zurückzukonvertieren. Somit kann diese DLL z.B. zum Austausch von Ressourcen-Informationen bei einer Audio-Video-Kommunikation verwendet werden, um zu ermitteln, welche Ressourcen ein Kommunikationspartner für eine Kommunikation mitbringt.

3.3. Auslesen von Ressourcen-Informationen

Die Hardware- und Softwareinformationen werden aus der Registry des entsprechenden Windows-Systems (vgl. Punkt 1.1.8), sowie über Methoden von DLLs, die direkt mit dem Kernel von Windows interagieren (vgl. Punkt 1.2) und aus dem SMBIOS (vgl. Punkt 1.3) gelesen.
Folgende Informationen über einen Rechner können ausgelesen werden:

- Installierte Hardwarekomponenten, samt Gerätestatus, Treiberversion, Treiberdatum, Hersteller etc. (analog zum Gerätemanager von Windows)
- Primäre Grafikkarte, inklusive z.B. Speichergröße und BIOS-Datum
- Installierte Software (analog dem Programm Software der Systemsteuerung)
- Laufwerke, inklusive freiem, gesamtem Speicher und Laufwerkstyp
- Maximale unterstütze Auflösungen der Anzeigegeräte bei bestimmter Pixelzahl und horizontaler Bildwiederholfrequenz
- Multimediageräte, wie z.B. Audio- und Videocodecs, Audiogeräte und Videoaufnahmegeräte (analog dem Programm Multimedia der Systemsteuerung unter dem Register Hardware)
- Lokale Netzwerkfreigaben, samt Pfad und Untersuchung auf fehlende Passwortbeschränkung (Untersuchung auf Passwortbeschränkung nur unter Windows 95/98/ME)
- Alle Prozessoren, samt Bezeichnung und Takt
- Routingtabelle (nicht unter Windows 95), (analog zum Programm route.exe)
- Freier und gesamter Speicher des Arbeitsspeichers und der Auslagerungsdatei
- Netzwerk- und DFÜ-Verbindungen über TCP/IP, samt Netzwerkinterface, Verbindungsname (nur Windows 2000/XP), MAC-Adresse (nicht Windows 95), max. Übertragungsgeschwindigkeit (nicht Windows 95), IP-Adresse, Subnetz-Maske, Gateway, DNS, DCHP (teilweise unter dem Programm Netzwerk- und DFÜ-Verbindungen der Systemsteuerung zu finden)
- Windows-Version

- SMBIOS:
    - Allgemein (z.B. BIOS-Name, -ID, -Version, -Datum)
    - Unterstützte Sprachen
    - Prozessoren, inklusive z.B. ID, interne und externe Geschwindigkeit, Socket, Spannung
    - Prozessor-Cache, samt Socket, Größe, Assoziativität
    - Speichermodule, inklusive Bank, Typ, Größe, Geschwindigkeit
    - Speichercontroller, samt Fehlererkennungsmethode und Fehlerbeseitungsfähigkeit
    - Onboard-Geräte
    - Systemboards, wie Motherboards, inklusive Modell, Version, Hersteller
    - Ports, wie USB oder PS/2, samt Bezeichnung, interne und externe Verbindung, Typ
    - Slots, wie PCI und AGP, inklusive Bezeichnung, Typ, Datenbusweite, aktuelle Verwendung
    - Physikalische Speicherfelder (die Speicherbänke des Arbeitsspeichers bilden z.B. zusammen ein Speicherfeld), samt Position, Art des Speicherfeldes, maximale Größe, Anzahl der Speichergeräte, die zum Speicherfeld gehören
    - Speichergeräte des Speicherfeldes, inklusive z.B. Position, Bank, Größe, Form Faktor, Typ
    - Chassis, samt Hersteller, Modell, Version, Seriennummer
    - System, samt Hersteller, Modell, Version, Seriennummer

3.4. Speicherformat von Ressourcen-Informationen

Wie unter Punkt 3.2.3 beschrieben, exportiert die Ressourcen-Info-DLL Methoden, die Ressourcen-Informationen als String zurückliefern können. Die Daten der Ressourcen-Informationen in diesem String sind im XML-Format gespeichert.

XML (Extensible Markup Language) [W3C98] wird als Universallösung aller Software-Integrations-Probleme gesehen. Immer mehr Hersteller integrieren XML in ihre Produkte. Somit können Daten zwischen verschiedenen Anwendungen in einem einheitlichen Format, dem XML-Format, ausgetauscht werden [Box01]. XML findet ebenso immer mehr Anwendung in Datenbanken, indem diese den Import und Export von XML-Dokumenten unterstützen oder Datenbanken direkt als XML-Datenbanken implementiert werden.

XML ist eine Teilmenge von SGML (Standard Generalized Markup Language) [Gol01] und soll HTML (Hypertext Markup Language) [W3C99], was eine Anwendung von SGML ist, als traditionelle Sprache im Internet bzw. Intranet ablösen[1]. HTML selber ist nur eine Sprache zur Auszeichnung von Dokumenten, wobei z.B. <*H1*> bis <*H6*> Überschriften und <*B*> eine Fettschrift definieren. XML dagegen stellt nur Vorschriften bereit, mit denen eine Auszeichnungssprache definiert werden kann. HTML besitzt also Tags[2] (z.B. <a>) und Attribute (*href* bei <a href=“ “>), die fest vorgegeben sind, wobei diese bei XML frei definiert werden können [Mac97].

In Abbildung 19 ist das Grundgerüst jeder Ressourcen-Information im XML-Format dargestellt. Die roten Passagen sind individuelle Informationen der jeweiligen Ressourcen-Information.

```
<RessourcenInformation>
  <ID>ID der Res-Info</ID>
  ...
  Daten der Res-Info
  ...
</RessourcenInformation>
```

Abbildung 19 : Darstellung des Grundgerüstes der Ressourcen-Informationen im XML-Format

[1] Nähere Informationen zu XML, SGML und HTML sind unter den angegebenen Quellenangaben zu finden.

[2] Bezeichnung aus HTML

In Abbildung 20 ist ein Beispiel der Ressourcen-Information *Windows-Version* im XML-Format dargestellt

```
<RessourcenInformation>
  <ID>1</ID>
  <fVersionstring>Windows 2000 Professional</fVersionstring>
  <fCSD>Service Pack 2</fCSD>
  <fBuild>2195</fBuild>
</RessourcenInformation>
```

Abbildung 20 : Darstellung eines Beispiels der Ressourcen-Information *Windows-Version* im XML-Format

Das Clientprogramm bietet die Möglichkeit, Ressourcen-Informationen mehrerer Rechner in eine Datei zu speichern. Diese Datei entspricht ebenfalls dem XML-Format, deren Grundgerüst in Abbildung 21 dargestellt ist.

```
<RessourcenAbfrage>
  <Server>
    <ServerID>ID des Servers</ServerID>
    <ServerIP>IP des Servers</ServerIP>
    <ServerPort>Port des Servers</ServerPort>
      <RessourcenInformation>
        <ID>ID der Res-Info</ID>
        ...
        Daten der Res-Info
        ...
      </RessourcenInformation>
      ...
      weitere Ressourcen-Informationen
      ...
  </Server>
  ...
  weitere Server-Einträge
  ...
</RessourcenAbfrage>
```

Abbildung 21 : Darstellung des Grundgerüstes der Ressourcen-Informationen mehrerer Rechner

3.5. Kommunikationsprotokoll

Die Kommunikation wird über TCP/IP realisiert. Da die Kommunikation verschlüsselt wird, ist TCP erforderlich, da es folgende Vorteile bietet:

- Es findet ein Verbindungsaufbau und -abbau statt.
- Die Datenübertragung erfolgt mit Fehlerkontrolle und Flusssteuerung.

Durch diese von TCP bereitgestellten Mechanismen werden folgende Probleme behandelt:

- Paketverlust
- Duplikate
- Reihenfolgevertauschung der Pakete
- Datenverluste infolge fehlender Empfangspuffer

Das UDP-Protokoll bietet diese Mechanismen nicht an. Sie müssten für das Ressourcen-Abfrage-System selber implementiert werden.

3.6. Kommunikationsverschlüsselung

Um ein optimales Maß an Sicherheit und Performance zu gewährleisten, werden die in Kapitel zwei beschriebenen Algorithmen, wie in Kapitel zwei erläutert, miteinander kombiniert, wie es z.B. auch bei SSL vollzogen wird.
Auf Zertifikate wird dabei verzichtet, da offizielle Zertifikate für jeden einzelnen Server, bei einer Zertifizierungsstelle vor Ort, angemeldet werden müssten. Um dies zu umgehen, wäre eine eigene Zertifizierungshierarchie notwendig. Jedoch besteht in diesem Fall das Problem, dass der öffentliche Schlüssel der Zertifizierungsstelle auf sicherem Weg zum Serverprogramm gelangen muss. Schlüssel öffentlicher Zertifizierungsstellen könnten, wie es die Programmierer diverser Webbrowser vollziehen, in den Quelltext integriert werden, was bei einer eigenen Hierarchie nicht geht, da in diesem Fall das Ressourcen-Abfrage-System an diese Hierarchie gebunden wäre. Beim Einbinden des Systems in

eine andere Hierarchie müsste der Schlüssel im Quelltext geändert werden. Bei diesem Ressourcen-Abfrage-System wird der öffentliche Schlüssel des Clientprogramms dem Serverprogramm mitgegeben, womit ein sicheres Senden des öffentlichen Schlüssels und somit eine Verwendung von Zertifikaten nicht mehr notwendig ist (siehe Punkt 3.7).

Nach dem Aufbau der Verbindung zwischen Serverprogramm und Clientprogramm sendet das Clientprogramm einen verschlüsselten Text zum Serverprogramm. Dieser Text ist mit dem privaten Schlüssel des Clientprogramms mittels der asymmetrischen Verschlüsselungsmethode RSA verschlüsselt. Damit Änderungen an diesen Daten ausgeschlossen sind, wird von dem verschlüsselten Text der Hashwert[1] (mit Hilfe von SHA-1) ermittelt und dieser ebenfalls mit dem privaten Schlüssel mittels RSA verschlüsselt (digitale Signatur). Diese Daten können nur mit Hilfe des öffentlichen Schlüssels des Clientprogramms entschlüsselt werden, der dem Serverprogramm bekannt ist. Kann das Serverprogramm den Text entschlüsseln, so wurde der Text mit dem zugehörigen privaten Schlüssel verschlüsselt. Abbildung 22 verdeutlicht den Vorgang der Verschlüsselung, Abbildung 23 den Vorgang der Entschlüsselung nach dem Verbindungsaufbau.

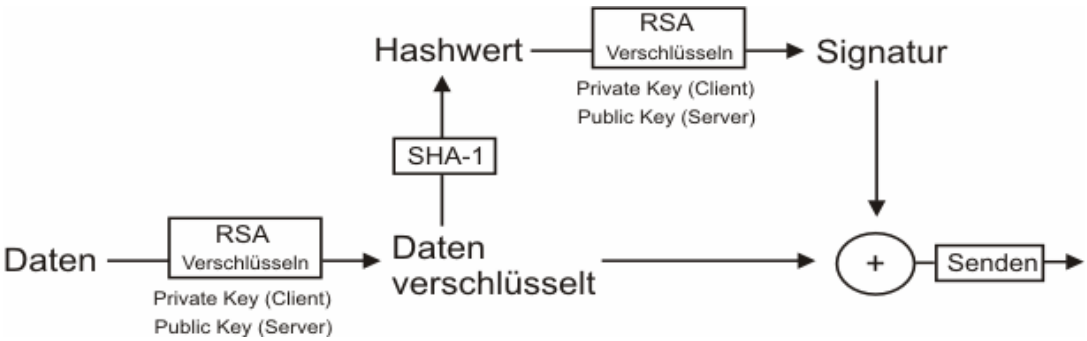


Abbildung 22 : Darstellung des Ablaufes der Verschlüsselung nach dem Verbindungsaufbau

[1] Hashwerte werden bei diesem Ressourcen-Abfrage-System immer durch SHA-1 generiert.

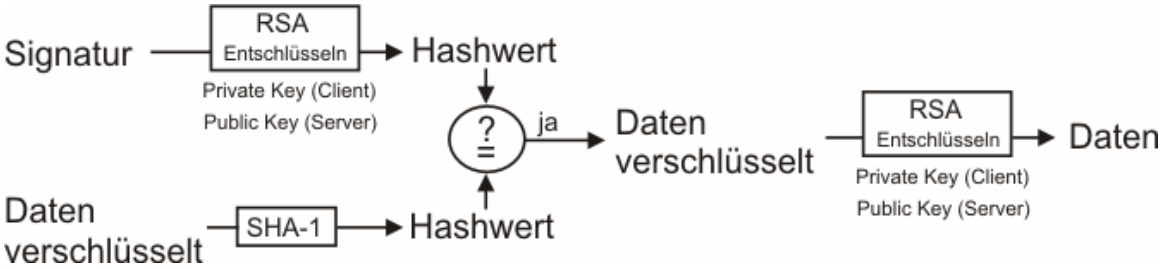


Abbildung 23 : Darstellung des Ablaufes der Entschlüsselung nach dem Verbindungsaufbau

Daraufhin generiert das Serverprogramm einen symmetrischen Schlüssel, der nur für die aktuelle Verbindung gültig ist und danach verworfen wird (genannt Sitzungsschlüssel). Der Sitzungsschlüssel wird mit dem öffentlichen Schlüssel des Clientprogramms mit Hilfe von RSA durch das Serverprogramm verschlüsselt. Vom verschlüsselten Sitzungsschlüssel wird der Hashwert bestimmt und dieser ebenfalls mit dem öffentlichen Schlüssel mit Hilfe von RSA verschlüsselt, um Verfälschungen an den Daten zu bemerken. Diese Daten werden zum Clientprogramm gesendet und können ebenfalls nur mit dem privaten Schlüssel des Clientprogramms entschlüsselt werden. Auch für den Fall, dass ein Dritter das Paket mit den Daten des verschlüsselten Textes, der am Anfang der Kommunikation vom Clientprogramm an das Serverprogramm gesendet wird, ebenfalls gelesen hat und versucht, mit Hilfe dieser Daten eine Kommunikation mit dem Serverprogramm zu erlangen, ist ausgeschlossen, dass er in Besitz eines Sitzungsschlüssels kommt, um mit dem Serverprogramm Daten auszutauschen. Ohne Kenntnis des privaten Schlüssels kann der Sitzungsschlüssel nicht entschlüsselt werden.

Nachdem Server- und Clientprogramm den Sitzungsschlüssel ausgetauscht haben, wird die weitere Kommunikation, also das Senden der Informationen über die Hard- und Softwareressourcen des Rechners, auf dem das Serverprogramm läuft, mit Hilfe dieses Schlüssels verschlüsselt. Dabei werden die zu sendenden Daten mit dem Sitzungsschlüssel mit Hilfe des symmetrischen Verschlüsselungsalgorithmus Rijndael (AES) verschlüsselt, der nicht so rechenintensiv wie RSA ist. Von den verschlüsselten Daten wird ein Hashwert erstellt und dieser ebenfalls mit Rijndael verschlüsselt. Da bei diesem Abfrage-System nur ein RSA-Schlüsselpaar vorhanden ist und nicht unterschieden werden muss, ob Serverprogramm oder Clientprogramm Daten signiert haben, ist es

nicht notwendig, den Hashwert mit RSA zu verschlüsseln, was rechenintensiver wäre. Auf eine Signatur könnte verzichtet werden, indem vom den unverschlüsselten Daten ein Hashwert erstellt würde. Jedoch müssten vor Überprüfung des Hashwertes die Daten entschlüsselt werden. Da normalerweise die Länge des Hashwertes geringer als die Länge des Datenwertes ist, wäre diese Variante der Überprüfung der Unverfälschtheit der Daten rechenintensiver, falls jemand versucht, das Serverprogramm durch das Senden vieler Pakete zu überlasten. Abbildung 24 verdeutlicht den Vorgang der Verschlüsselung, Abbildung 25 den der Entschlüsselung nach dem Austausch des Sitzungsschlüssels.

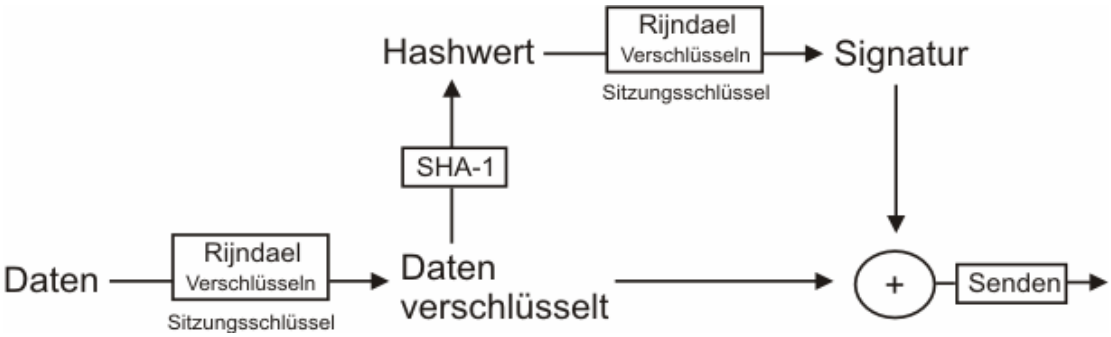


Abbildung 24 : Darstellung des Ablaufes der Verschlüsselung nach dem Austausch des Sitzungsschlüssels

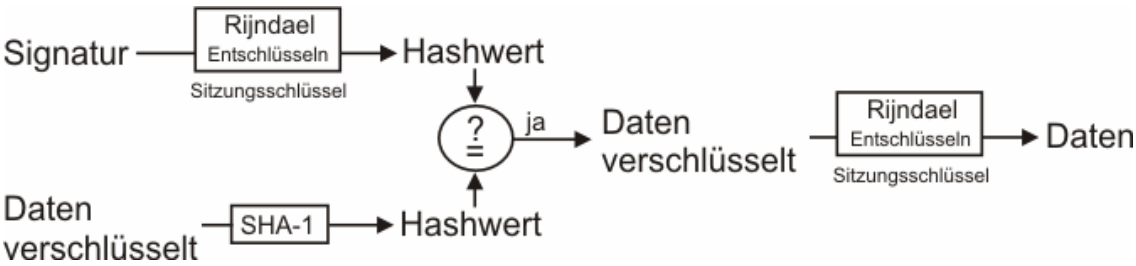


Abbildung 25 : Darstellung des Ablaufes der Entschlüsselung nach dem Austausch des Sitzungsschlüssels

3.7. Konfigurationsdateien

Unter Windows 2000 und XP gibt es Möglichkeiten, Daten vor den Augen anderer zu verbergen. So bietet z.B. die Registry dieser Windowsversionen die Möglichkeit, Zugriffsrechte auf jeden beliebigen Registry-Schlüssel zu setzen. Ebenso besteht diese Möglichkeit bei NTFS auf Dateiebene. Jedoch werden bei diesem Ressourcen-Abfrage-System auch die Windowsversionen 95,98 und ME einbezogen, bei denen Registry und Dateisystem keine Zugriffsbeschränkungen anbieten, die durch den Nutzer verwendet werden können. Da das Serverprogramm ohne interaktive Eingabe eines Passwortes gestartet werden soll und dennoch eine Zugriffsbeschränkung und ein sicheres Speichern der Konfigurationsdaten gewährleistet werden soll, wird ein so genannter Source-Key verwendet.
Dieser SourceKey ist ein Schlüssel, der mindestens eine Länge von 128 Bit aufweist. Er wird für die grundlegende Verschlüsselung der Konfigurationsdaten mit Hilfe der Rijndael-Verschlüsselungsmethode verwendet und ist im Quellcode von Client- und Serverprogramm verankert. Auf welche Art und Weise der SourceKey verwendet wird, wird anschließend bei der Beschreibung der jeweiligen Konfigurationsdateien erläutert. Für die grundlegende Verschlüsselung wurde Rijndael gewählt, da er ressourcenschonender als RSA ist und somit den Rechner beim Ver- und Entschlüsseln der Konfigurationsdaten weniger belastet.

3.7.1. ClientFile

Das ClientFile(*.cf) wird als erste Datei im Clientprogramm angelegt und enthält folgende Daten:

- Privater Schlüssel
- Öffentlicher Schlüssel
- Schlüssellänge
- Passwortabfrage beim Aufruf der Serververwaltung?
- Passwortabfrage beim Starten der Serverabfragen?

Als letztes Datum gibt der Nutzer noch ein Passwort ein. Dieses wird jedoch nicht in der Datei gespeichert, sondern wird für die Verschlüsselung der Daten verwendet.
Alle anderen Konfigurationsdateien des Abfrage-Systems sind von den Daten des ClientFile abhängig. Das ClientFile wird jedoch zum Betrieb des Server- bzw. Clientprogramms nur vom Clientprogramm benötigt.
Zuerst wird von den Daten ein Hashwert erzeugt. Danach werden die Daten mit Rijndael verschlüsselt. Als Schlüssel wird der SourceKey, mit dem Passwort kombiniert, verwendet. Diese verschlüsselten Daten, zusammen mit dem Hashwert, ergeben das ClientFile. Damit ist es trotz Kenntnis des SourceKey, die Entwickler des Programms haben, nicht möglich auf die Daten der Datei zu schließen.
Die Daten des ClientFile bilden die Grundlage für alle anderen Konfigurationsdateien. Nur mit Hilfe der Daten des ClientFile ist es möglich, diese Konfigurationsdateien anzulegen bzw. auszulesen.

3.7.2. ServerFile

Welche Serverprogramme und welche Ressourcen-Informationen das Clientprogramm abfragen soll, wird im ServerFile(*.sf) gespeichert. Für jeden Server-Eintrag werden folgende Daten in dieser Datei gespeichert:

- IP-Adresse des Rechners, auf dem das Serverprogramm läuft
- Portadresse, auf dem das Serverprogramm lauscht
- Welche Ressourcen-Informationen sollen abgefragt werden?
- Soll der Server abgefragt werden?

Das ServerFile ist vom ClientFile abhängig, d.h. ein ServerFile kann nur mit dem entsprechenden ClientFile entschlüsselt werden. Dies wird erreicht, indem die Daten des ServerFile mit dem öffentlichen Schlüssel des ClientFile mit Hilfe von RSA verschlüsselt werden. Von den unverschlüsselten Daten wird ein Hashwert erzeugt. Die verschlüsselten Daten und der Hashwert ergeben das ServerFile. Somit können die Daten nur mit Hilfe des privaten Schlüssels des ClientFile entschlüsselt werden.

3.7.3. Das Serverprogramm

Mit Hilfe des Clientprogramms kann das kompilierte Serverprogramm so verändert werden, dass es verschlüsselte Daten mit sich führt. Diese Daten werden mit dem SourceKey verschlüsselt und beinhalten folgende Werte:

- Öffentlicher Schlüssel, der im ClientFile gespeichert ist
- Schlüssellänge
- Hashwert des Passwortes, das im ClientFile gespeichert ist
- Hashwert des Zugriffspasswortes auf das Serverprogramm

Personen, die den SourceKey kennen, wie z.B. Entwickler dieses Abfrage-Systems, können mit diesen Daten nichts anfangen. Um die Kommunikation überwachen zu können, wird der private Schlüssel benötigt, da es nur mit ihm möglich ist, den Sitzungsschlüssel zu entschlüsseln. Des Weiteren lassen die Hashwerte der Passwörter nicht auf die Passwörter selber schließen. Auf welche Art und Weise das kompilierte Serverprogramm verändert wird, damit diese Daten enthalten sind, wird aus Sicherheitsgründen im Quelltext beschrieben.

3.7.4. ServerConfigFile

Die mit dem Serverprogramm gespeicherten Daten sind bei allen Serverprogrammen gleich, die vom Clientprogramm mit Hilfe des entsprechenden ClientFile abgefragt werden sollen. Jedoch gibt es auch Daten, die sich von Serverprogramm zu Serverprogramm unterscheiden können. Dies sind zum einen der Port, auf dem das Serverprogramm lauschen soll und zum anderen, ob das Serverprogramm beim Starten von Windows auch gestartet werden soll. Damit es auf einfache Art und Weise möglich ist, diese Daten bei einigen Serverprogrammen unterschiedlich zu halten, ohne jedes Mal das kompilierte Serverprogramm zu verändern, wird das ServerConfigFile(*.scf) verwendet.
Von den zu speichernden Daten wird ein Hashwert erstellt. Danach werden die Daten mit Hilfe von RSA und dem privaten Schlüssel des entsprechenden ClientFile verschlüsselt. Mit Hilfe des öffentlichen Schlüssels, der dem kompi-

lierten Serverprogramm mitgegeben wird, können diese Daten entschlüsselt werden. Hashwert und verschlüsselte Daten bilden das ServerConfigFile.

3.8. Ablauf zwischen Server- und Clientprogramm

Nachdem das Clientprogramm erfolgreich das ClientFile und ServerFile geladen und entschlüsselt hat und das Serverprogramm erfolgreich die Daten aus der eigenen kompilierten Datei und des ServerConfigFile gelesen hat, kann die Kommunikation zwischen Server- und Clientprogramm erfolgen.
Auf welche Art und Weise die Kommunikationsverschlüsselung initiiert wird, wurde unter Punkt 3.6 beschrieben. Dieser Abschnitt schließt bei dem Punkt an, nachdem der Sitzungsschlüssel beiden Kommunikationsparteien bekannt ist und die weitere Kommunikationsverschlüsselung mit Hilfe von Rijndael und des Sitzungsschlüssels vollzogen wird.
Nachdem der Sitzungsschlüssel übermittelt wurde, sendet das Clientprogramm dem Serverprogramm die Information, welche Ressourcen-Informationen übermittelt werden sollen. Daraufhin sendet das Serverprogramm den Namen des Rechners, auf dem es läuft, welcher für die Anzeige des Servereintrages im Baum des Clientprogramms verwendet wird. Das Clientprogramm quittiert den Empfang. Nachdem das Serverprogramm diese Quittung erhalten hat, lädt es die Ressourcen-Info-DLL dynamisch in den Speicher, liest mit Hilfe dieser DLL die Ressourcen-Informationen des Rechners aus und sendet sie fortlaufend in einer Schleife an das Clientprogramm. Jede Ressourcen-Information, die das Clientprogramm erhalten hat, quittiert es. Damit weiß das Serverprogramm, dass das Clientprogramm die Daten nicht nur erhalten hat, sondern dass diese Daten auch unverändert das Clientprogramm erreicht haben. Nachdem das Clientprogramm alle Ressourcen-Informationen erhalten und gespeichert hat, fügt es den Eintrag des entsprechenden Serverprogramms dem Baum, in dem alle Servereinträge angezeigt werden, hinzu und fährt mit der Abfrage des nächsten Serverprogramms fort. Abbildung 26 zeigt einen grafischen Überblick über den Ablauf zwischen Server- und Clientprogramm.

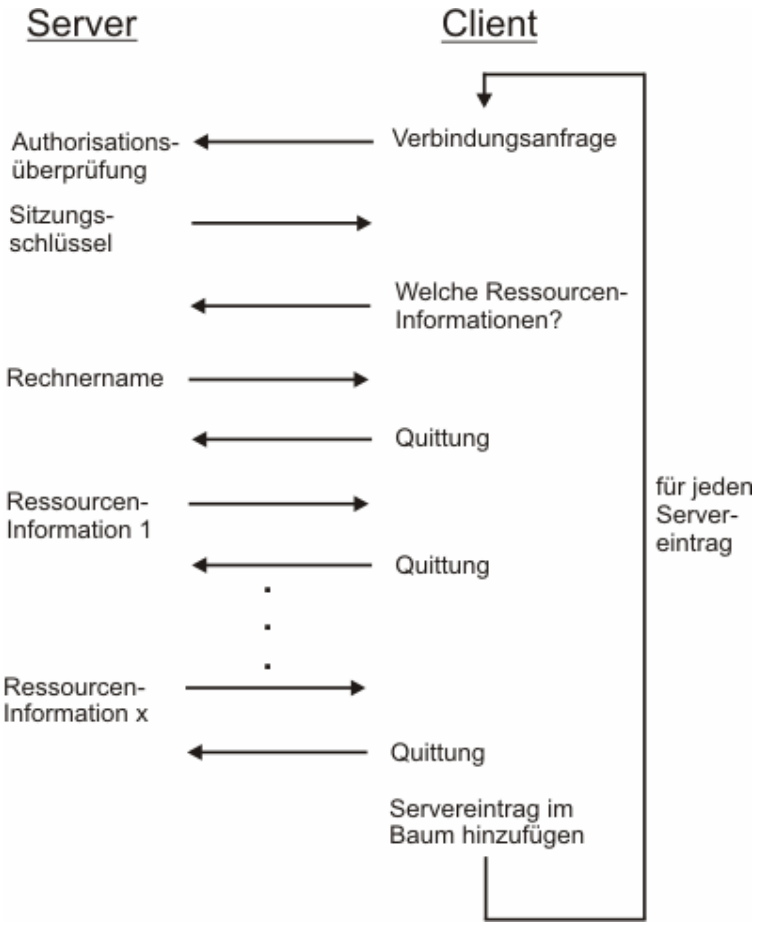


Abbildung 26 : Darstellung des Kommunikations- und Arbeitsablaufes zwischen Server- und Clientprogramm

Sobald ein Schritt dieses Ablaufes nicht korrekt ausgeführt werden kann oder manipulierte Daten (Signatur stimmt nicht überein) einen Kommunikationspartner erreichen, wird die Verbindung zum jeweiligen Kommunikationspartner beendet und muss neu aufgebaut werden.

3.9. Bedienung der Applikationen

3.9.1. Das Clientprogramm

Nach dem Starten des Clientprogramms erscheint das Hauptfenster des Clientprogramms, dargestellt in Abbildung 27.

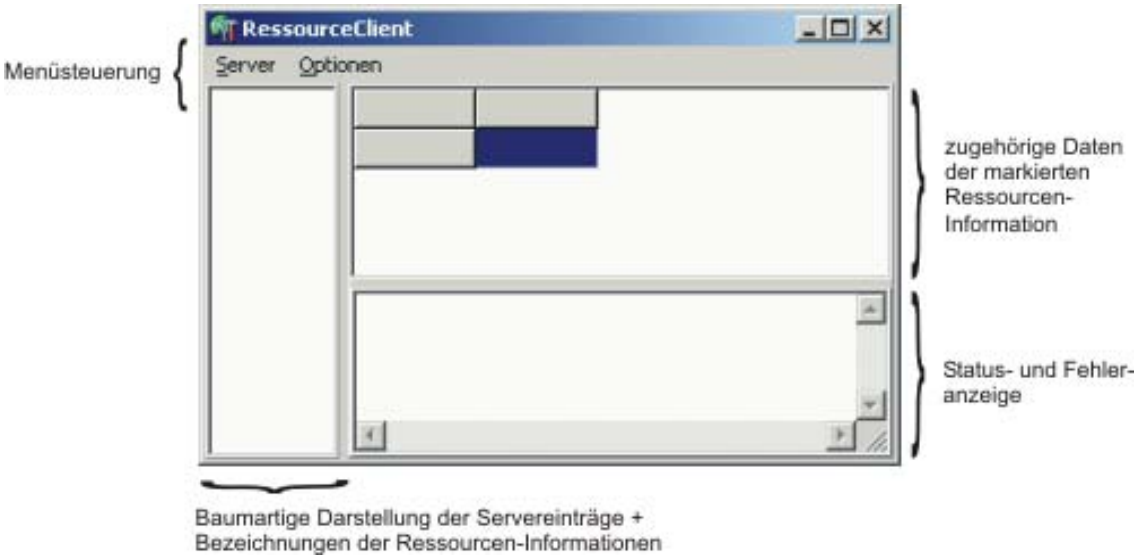


Abbildung 27 : grafische Darstellung des Hauptfensters des Clientprogramms

Unter dem Menüpunkt „*Server*“ sind folgende Punkte zu finden, von denen beim erstmaligen Start des Clientprogramms nur die mit * markierten Punkte zur Wahl stehen:

- *„Ressourcen-Abfrage starten“* (startet das Abfragen der Ressourcen-Informationen der einzelnen Serverprogramme)
- * *„Ressourcen-Abfrage laden“* (lädt gespeicherte abgefragte Ressourcen-Informationen aus einer Datei)
- *„Ressourcen-Abfrage speichern“* (speichert abgefragte Ressourcen-Informationen in eine Datei)
- *„Ressourcen-Abfrage löschen“* (löscht die aktuell abgefragten Ressourcen-Informationen aus dem Baum und dem Speicher)
- * *„Beenden“* (beendet das Clientprogramm)

Der Menüpunkt „*Optionen*“ beinhaltet folgende Einträge, von denen beim erstmaligen Starten des Clientprogramms ebenfalls nur die mit * markierten Einträge wählbar sind:

- * „*ClientFile erzeugen / Passwort ändern/setzen*“ (ruft ein Fenster auf, auf dem ein ClientFile angelegt werden kann und das Passwort gesetzt bzw. geändert werden kann)
- * „*ClientFile laden*“ (lädt ein gespeichertes ClientFile)
- „*ServerFile laden*“ (lädt ein gespeichertes ServerFile)
- „*Client-Optionen*“ (ruft ein Fenster auf, auf dem festgelegt werden kann, wann das Passwort des ClientFile abgefragt werden soll)
- „*Client-Key in Server schreiben*“ (verändert die kompilierte Datei des Serverprogramms so, dass sie die Daten aus Punkt 3.7.3 mit sich führt)
- „*Serververwaltung*“ (ruft ein Fenster auf, auf dem die Einträge des ServerFile verändert werden können und ein ServerConfigFile angelegt werden kann)

3.9.1.1. Anlegen eines ClientFile

Zuerst wird mit einem Klick auf den Eintrag „*ClientFile erzeugen / Passwort ändern/setzen*“ unter dem Menüpunkt „*Optionen*“ das Fenster zum Anlegen eines ClientFile bzw. zum Setzen des Passwortes aufgerufen. Ein Screenshot dieses Fenster ist in Abbildung 28 dargestellt.

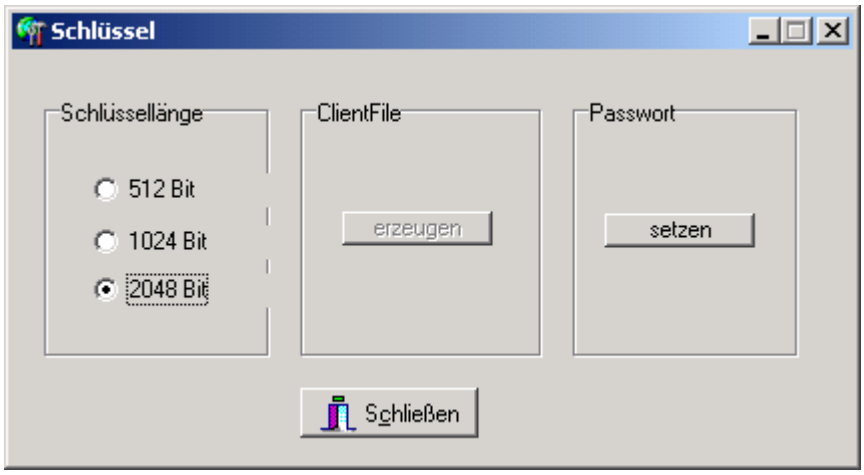


Abbildung 28 : Screenshot des Fensters zum Anlegen des ClientFile bzw. zum Setzen/Ändern des Passworts

Durch einen Klick auf den Button *„setzen“* des Bereiches *„Passwort“* wird ein Passwortdialog aufgerufen, auf dem ein neues Passwort eingestellt wird, das zum Verschlüsseln des ClientFile sowie zum Zugriffsschutz auf bestimmte Bereiche des Clientprogramms verwendet wird. Das Passwort darf nicht leer sein und keine Leerzeichen enthalten.
Nach dem erfolgreichen Einstellen des Passworts ist der Button *„erzeugen“* des Bereiches *„ClientFile“* aktiv. Es kann also ein neues ClientFile erzeugt werden. Im Bereich „*Schlüssellänge“* wird die Schlüssellänge des RSA-Schlüssels festgelegt, der im ClientFile gespeichert wird. Da ein Schlüssel einer Länge von 512 Bit bereits erfolgreich kompromittiert wurde (vgl. Punkt 2.3.2.1), wird eine Schlüssellänge von 1024 Bit, oder falls der Schlüssel längere Zeit verwendet werden soll, 2048 Bit, empfohlen. Durch einen Klick auf den Button *„erzeugen“* wird das ClientFile erzeugt. Zuerst wird gefragt, ob ein anderes Passwort als das aktuelle Passwort verwendet werden soll. So kann ein neues ClientFile generiert werden, ohne das bestehende Passwort ändern zu müssen. Soll ein anderes Passwort verwendet werden, wird der eben verwendete Passwortdialog aufgerufen. Nach der Passworteinstellung wird der RSA-Schlüssel erzeugt. Danach werden der Speicherort und der Dateiname des zu speichernden ClientFile angegeben. Das ClientFile erhält automatisch die Endung *.cf*. Anschließend wird gefragt, ob das eben gespeicherte ClientFile auch geladen werden soll. Wird diese Frage positiv beantwortet, wird gefragt, ob dieses ClientFile als Standard gesetzt werden soll. Das bedeutet, dass der Speicherort und der Dateiname im HKCU-Abschnitt der Registry gespeichert werden und das ClientFile beim nächsten Starten des Clientprogramms automatisch geladen wird. Danach kann dieses Fenster geschlossen werden.
Nach dem erfolgreichen Anlegen und Laden des ClientFile können die folgenden, vorher inaktiven Einträge des Menüpunktes *„Optionen“*, gewählt werden:

- *„ServerFile laden“*
- *„Client-Optionen“*
- *„Client-Key in Server schreiben“*
- *„Serververwaltung“*

3.9.1.2. Client-Optionen

Ein Klick auf diesen Eintrag ruft ein Fenster auf, auf dem eingestellt werden kann, bei welchen Bereichen des Clientprogramms das Passwort, das zur Verschlüsselung des ClientFile verwendet wird, abgefragt werden soll. Ein Screenshot dieses Fensters ist in Abbildung 29 abgebildet.

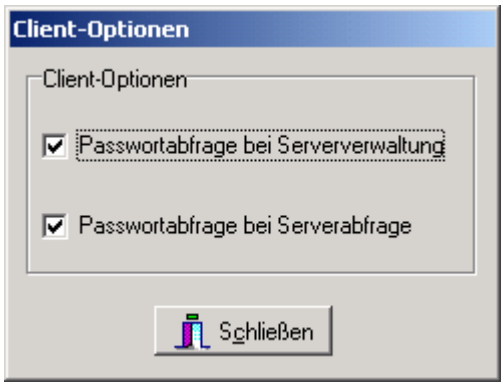


Abbildung 29 : Screenshot des Fensters Client-Optionen

Bevor jedoch dieses Fenster angezeigt wird, muss dieses Passwort eingegeben werden. Diese Passwortabfrage kann nicht deaktiviert werden, sodass Änderungen an den Passwortabfragen ohne Kenntnis des Passwortes nicht vollzogen werden können. Nachdem das korrekte Passwort eingegeben wurde, können Einstellungen der Passwortabfragen verändert werden. So kann gewählt werden, ob eine Passwortabfrage, vor dem Anzeigen des Fensters zur Verwaltung des ServerFile bzw. ServerConfigFile oder vor dem Starten der Abfragen der Serverprogramme, stattfinden soll.

3.9.1.3. Anlegen eines ServerFile und ServerConfigFile

Durch einen Klick auf den Eintrag *„Serververwaltung“* des Menüpunktes *„Optionen“* wird das Fenster zum Verwalten der Servereinträge im ServerFile bzw. zum Anlegen des ServerConfigFile aufgerufen. Dieser Menüeintrag kann erst gewählt werden, nachdem ein ClientFile erfolgreich geladen werden konnte.
Ist die Passwortabfrage beim Aufruf der Serververwaltung aktiv, muss erst das Passwort des ClientFile eingegeben werden, um diesen Bereich betreten zu können. Nach erfolgreichem Eingeben des Passworts erscheint das Fenster.
Wurde noch kein ServerFile geladen erscheint eine Abfrage, ob ein ServerFile geladen werden soll. Wurde noch kein ServerFile angelegt bzw. gespeichert, muss die Abfrage negativ beantwortet werden. Danach erfolgt eine Abfrage, ob ein neues ServerFile angelegt werden soll. Wird diese Abfrage positiv beantwortet, erscheint ein Dialog, in dem der Speicherort und der Dateiname des ServerFile angegeben werden soll. Die Datei erhält automatisch die Endung *.sf*. Danach erfolgt analog wie beim Anlegen des ClientFile die Frage, ob das ServerFile als Standard gesetzt werden soll, also, ob es beim nächsten Starten des Clientprogramms automatisch geladen werden soll.
Anschließend wird das Fenster zur Verwaltung der Einträge des ServerFile bzw. zum Anlegen des ServerConfigFile angezeigt, welches in Abbildung 30 dargestellt ist.

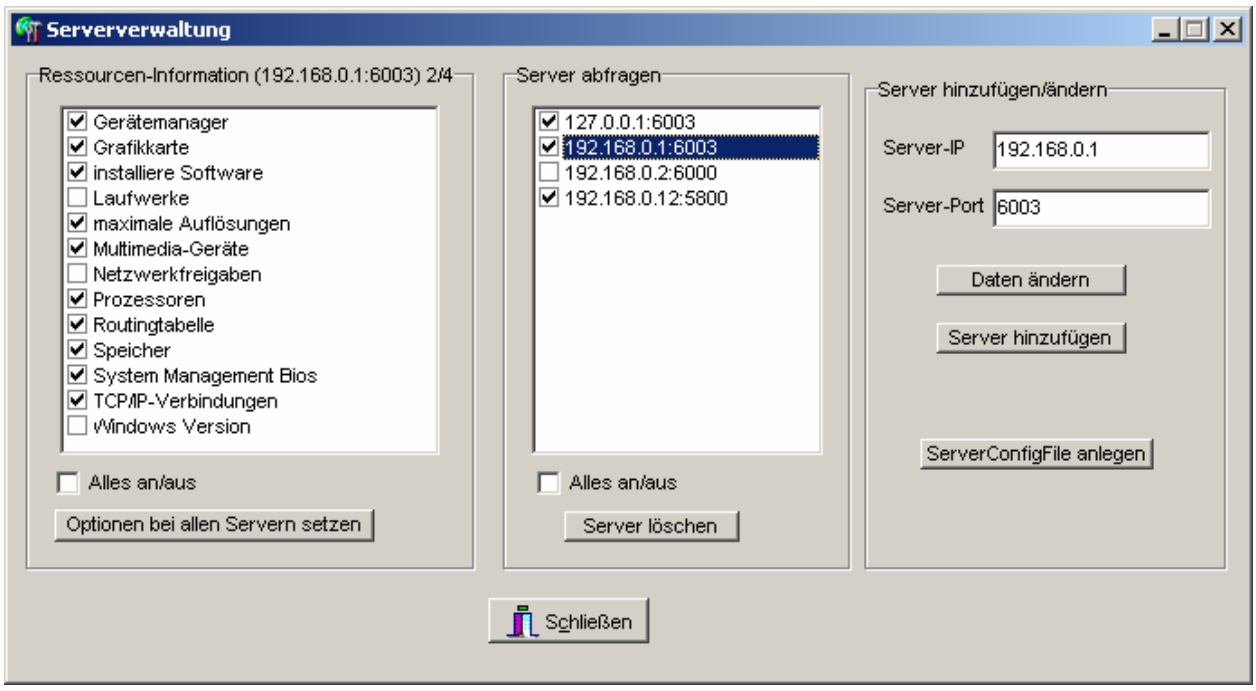


Abbildung 30 : Screenshot des Fensters der Serververwaltung

Der Screenshot aus Abbildung 30 enthält bereits Servereinträge, wobei der 2. Eintrag markiert ist. Sind noch keine Einträge vorhanden, ist der Bereich *„Ressourcen-Information“* deaktiviert.

Im Bereich *„Server hinzufügen/ändern“* können neue Servereinträge hinzugefügt werden, indem im Feld *„Server-IP“* die IP-Adresse des Serverprogramms und im Feld *„Server-Port“* der Port eingetragen wird, auf dem das Serverprogramm lauscht. Anschließend werden die Daten durch einen Klick auf den Button *„Server hinzufügen“* im ServerFile gespeichert und der Servereintrag in der Form *„Server-IP:Server-Port“* im Bereich *„Server abfragen“* eingetragen.

Wird der eben erstellte Eintrag im Bereich *„Server abfragen“* markiert, werden die Einträge des Bereiches *„Ressourcen-Information“* sichtbar. Die Daten des Eintrages werden dabei ebenfalls in die Felder des Bereiches *„Server hinzufügen/ändern“* eingetragen, so dass die Daten geändert werden können. Mit einem Klick auf den Button *„Daten ändern“* werden diese Änderungen aktiviert. Der Haken neben dem Servereintrag im Bereich *„Server abfragen“* charakterisiert, ob der eingetragene Server beim Abfragen der Ressourcen-Informationen abgefragt werden soll. Ein Klick auf den Button *„Server löschen“* entfernt den markierten Servereintrag. Durch einen Klick auf *„Alles an/aus“* in den Bereichen *„Ressourcen-Information“* und *„Server abfragen“* werden alle Einträge aktiviert bzw. deaktiviert.

Der Bereich *„Ressourcen-Information“* enthält die Bezeichnungen aller Ressourcen-Informationen, die abgefragt werden können. Welche Ressourcen-Informationen abgefragt werden sollen, wird für jeden Servereintrag individuell gespeichert. Durch einen Klick auf einen Servereintrag im Bereich *„Server abfragen“* wird im Bereich *„Ressourcen-Information“* angezeigt, welche Informationen speziell bei diesem Server abgefragt werden sollen. Welcher Servereintrag aktiv ist, ist an der Markierung im Bereich *„Server anfragen“* und an der Bezeichnung des Bereiches *„Ressourcen-Information“* zu sehen, wie in Abbildung 30 dargestellt. Durch einen Klick auf den Button *„Optionen bei allen Servern setzen“* wird die aktuelle Einstellung der Abfrage der Ressourcen-Informationen bei jedem Servereintrag gesetzt, so dass alle Servereinträge die gleiche Einstellung besitzen.
Das ServerConfigFile wird durch einen Klick auf den Button *„ServerConfigFile anlegen“* erstellt. Als erstes wird der Port abgefragt, auf dem das Serverprogramm lauschen soll. Danach wird gefragt, ob das Serverprogramm automatisch beim Starten von Windows geladen werden soll. Anschließend wird der Speicherort und der Dateiname des ServerConfigFile bestimmt. Die Datei erhält automatisch die Endung *.scf*.

3.9.1.4. Client-Key in Server schreiben

Durch einen Klick auf diesen Eintrag werden der ausführbaren Datei des Serverprogramms entsprechende Daten des ClientFile mitgegeben (vgl. Punkt 3.7.3). Ohne diese Daten kann das Serverprogramm nicht betrieben und das ServerConfigFile nicht entschlüsselt werden.
Als erstes wird die ausführbare Datei des Serverprogramms gewählt. Danach muss ein Passwort eingegeben werden, welches beim Anzeigen des Hauptfensters und beim Beenden des Serverprogramms abgefragt wird. Falls dem Serverprogramm schon Daten mitgegeben wurden, wird gefragt, ob diese Daten ersetzt werden sollen. Falls diese Frage positiv beantwortet wird und sich das Passwort des ClientFile, aus dem das Serverprogramm die Daten bezogen hat, vom aktuellen Passwort das ClientFile unterscheidet, muss dieses Passwort eingegeben werden. Andernfalls erfolgt diese Abfrage nicht. Anschließend

werden die Daten dem Serverprogramm mitgegeben.
Zusammen mit dem entsprechenden ServerConfigFile ist das Serverprogramm lauffähig.

3.9.1.5. Ressourcen-Abfrage

Nachdem alle Konfigurationsdateien erstellt wurden und die entsprechenden Daten des ClientFile dem Serverprogramm mitgegeben wurden, können die gewählten Ressourcen-Informationen der Servereinträge des ServerFile abgefragt werden. Dies geschieht durch einen Klick auf dem Eintrag *„Ressourcen-Abfrage“ starten* des Menüpunktes *„Server“*. Danach werden die Ressourcen-Informationen der jeweiligen Rechner hintereinander abgefragt und in die baumartige Darstellung auf der linken Seite des Clientfensters eingetragen. Während der Kommunikation wird der Status der Übertragung im unteren Bereich des Clientfensters ausgegeben. Nachdem alle gewählten Servereinträge abgefragt wurden, können die zugehörigen Daten der gewählten Ressourcen-Information in der tabellenartigen Ansicht auf der rechten Seite des Clientfensters angezeigt werden. Screenshots folgen im Kapitel fünf, in dem Beispielabfragen gezeigt werden.
Durch einen Klick auf dem Eintrag *„Ressourcen-Abfrage speichern“* kann eine komplette Ressourcen-Abfrage samt abgefragter Daten in eine Datei gespeichert werden. Mit Hilfe des Eintrages *„Ressourcen-Abfrage laden“* kann die gespeicherte Abfrage wieder geladen werden. Das Löschen der aktuellen Ressourcen-Abfrage aus dem Speicher geschieht durch einen Klick auf den Eintrag *„Ressourcen-Abfrage löschen“*.

3.9.2. Das Serverprogramm

Das Serverprogramm existiert in zwei Varianten:

- als eigenständige Applikation
- als Dienst für Windows 2000/XP

Das Serverprogramm, als eigenständige Applikation, ist unter Windows 95/98/ ME/2000/XP lauffähig, als Dienst nur unter Windows 2000/XP.
Bevor das Serverprogramm lauffähig ist, müssen ihm die entsprechenden Daten des ClientFile mitgegeben werden (vgl. Punkt 3.7.3). Des Weiteren benötigt das Serverprogramm noch ein ServerConfigFile (vgl. Punkt 3.7.4). Standardmäßig lädt das Serverprogramm beim Starten das ServerConfigFile mit dem Namen *serverconfig.scf*, jedoch kann als erster Kommandozeilenparameter beim Aufruf ein anderer Dateiname übergeben werden. Ebenso wird die Ressourcen-Info-DLL zum Auslesen der Ressourcen-Informationen benötigt.
Erst nachdem das Serverprogramm die mitgegebenen Daten des ClientFile und das ServerConfigFile erfolgreich geladen und entschlüsselt hat, können Ressourcen-Informationen abgefragt werden. Ansonsten gibt das Serverprogramm eine entsprechende Fehlermeldung aus und beendet sich.
Das Serverprogramm als eigenständige Applikation minimiert sich nach erfolgreichem Start und platziert in der Tray-Leiste ein Icon, welches ein Popup-Menü besitzt. Das Popup-Menü enthält zwei Einträge:

- *„Hauptfenster anzeigen“*
- *„Server beenden“*

Erst nachdem das richtige Server-Passwort eingegeben wurde, wird die gewählte Aktion des Popup-Menüs ausgeführt.
Sobald ein Clientprogramm eine Verbindung aufbaut, wird dies im Hauptfenster des Serverprogramms angezeigt. Ebenso wird der Ablauf der Kommunikation protokolliert. Abbildung 31 zeigt einen Screenshot des Serverprogramms als eigenständige Applikation.
Das Serverprogramm als Dienst besitzt keine visuellen Anzeigemöglichkeiten.

Es kann nur mit Hilfe des Dienstkontroll-Managers gestartet und beendet werden. Fehlermeldungen während der Kommunikation mit dem Clientprogramm werden als Warnung im Fehlerprotokoll von Windows ausgegeben. Wie auch das Serverprogramm als eigenständige Applikation gibt der Dienst eine Fehlermeldung beim Starten aus, falls die benötigten Daten nicht geladen werden können und beendet sich.

Abbildung 31 : Screenshot des Serverprogramms als eigenständige Applikation

4. Implementierung

Um die Konzeption umzusetzen, wurde die Anwendungsentwicklungsumgebung Delphi in der Version 5, die als Sprache Object Pascal verwendet, von der Firma Borland eingesetzt [Bor02].

4.1. Ressourcen-Informationen

Jede Ressourcen-Information wird in einer eigenen Klasse gekapselt. Alle Ressourcen-Informations-Klassen werden von einer Basisklasse namens *TBasisKlasse* abgeleitet, welche selber von der Klasse *TObject* abgeleitet ist. *TObject* ist der Ausgangspunkt der Klassenhierarchie und ist Vorfahr aller VCL-Objekte und Komponenten. *TBasisKlasse* beinhaltet Methodendeklarationen, die als *virtual* und *abstract* definiert sind. Die Direktive *virtual* bedeutet, dass die Methode in einer abgeleiteten Klasse überschrieben werden kann. Die Direktive *abstract* bedeutet, dass die Methode nicht in der Klasse implementiert wird, in der sie deklariert wurde. Die Implementierung erfolgt in einer der abgeleiteten Klassen, indem die Methode mit der Direktive *override* deklariert wird, da sie überschrieben wird. Das heißt, dass jede abgeleitete Klasse von *TBasisKlasse*, also jede Ressourcen-Informations-Klasse, die Implementierung dieser Methoden selber übernimmt. Diese Methoden bieten unter anderem folgende Funktionalitäten für jede Ressourcen-Informations-Klasse:

- Ressourcen-Analyse durchführen
- Ressourcen-Information als String zurückgeben
- Ressourcen-Information, die als String vorliegt, importieren
- Name bzw. ID der Ressourcen-Information zurückliefern
- Baumartige Anzeige der Ressourcen-Information
- Tabellenartige Anzeige der Daten der Ressourcen-Information

Der Vorteil dieser Verfahrensweise liegt darin, dass einer Instanz von *TBasisKlasse* zur Laufzeit eine abgeleitete Klasse (also eine Ressourcen-Informations-Klasse) zugewiesen werden kann. Diese Instanz kann dann die

eben aufgezählten Methoden aufrufen, deren Implementierungen in der abgeleiteten Klasse erfolgt ist. Somit liefern diese Methoden das gleiche Datenformat zurück, allerdings mit den Daten der jeweils abgeleiteten Klasse. Diese Methodik wird als Polymorphie bezeichnet, da sich hinter dem gleichen Interface unterschiedliche Implementierungen einer Methode verbergen [Dob00]. Abbildung 32 zeigt den Vorteil der Polymorphie an einem Beispiel für Delphi.

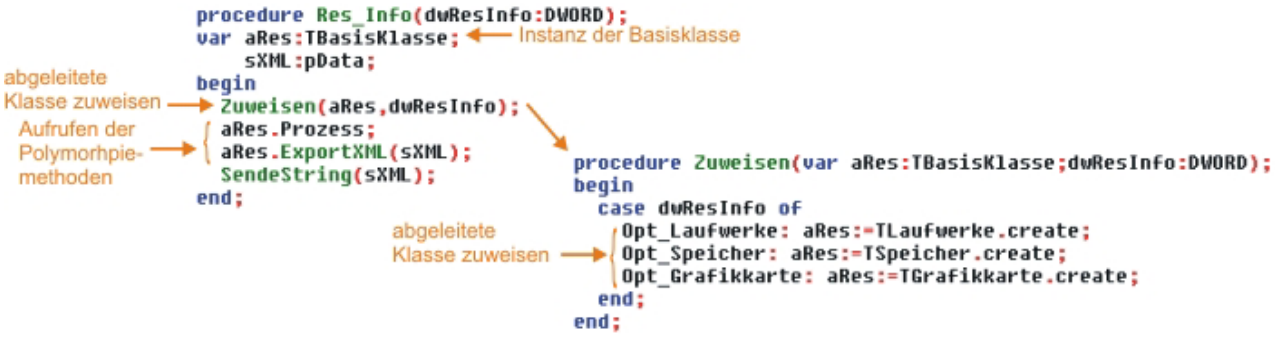


Abbildung 32 : Darstellung eines Beispiels zum Verdeutlichen der Vorteile der Polymorphie

Ohne Verwendung der Polymorphie müsste jede Instanz einer Ressourcen-Informations-Klasse einzeln deklariert werden. Ebenso müsste jede Methode einzeln mit der jeweiligen Instanz aufgerufen werden.

4.2. Ressourcen-Informationen im XML-Format

4.2.1. Verwendete Software

Als XML-Verarbeitungsoftware wird *Open XML* [Koe02] verwendet. *Open XML* wurde in Delphi entwickelt und implementiert unter anderem das DOM 1 (Document Object Model) [W3C98a] und DOM 2 [W3C00][1].
Das DOM ist eine API für HTML- und XML-Dokumente. Es liefert einen Satz von Schnittstellen, mit denen es möglich ist, Dokumente zu erstellen, in ihnen zu navigieren sowie Elemente und Inhalte hinzuzufügen, zu verändern und zu löschen. Das HTML- oder XML-Dokument wird dabei als Baum repräsentiert. In Abbildung 33 ist im oberen Bereich ein XML-Dokument und im unteren Bereich die Abbildung des XML-Dokuments nach DOM, samt Begriffswelt des DOM, dargestellt [W3C00][Box01].

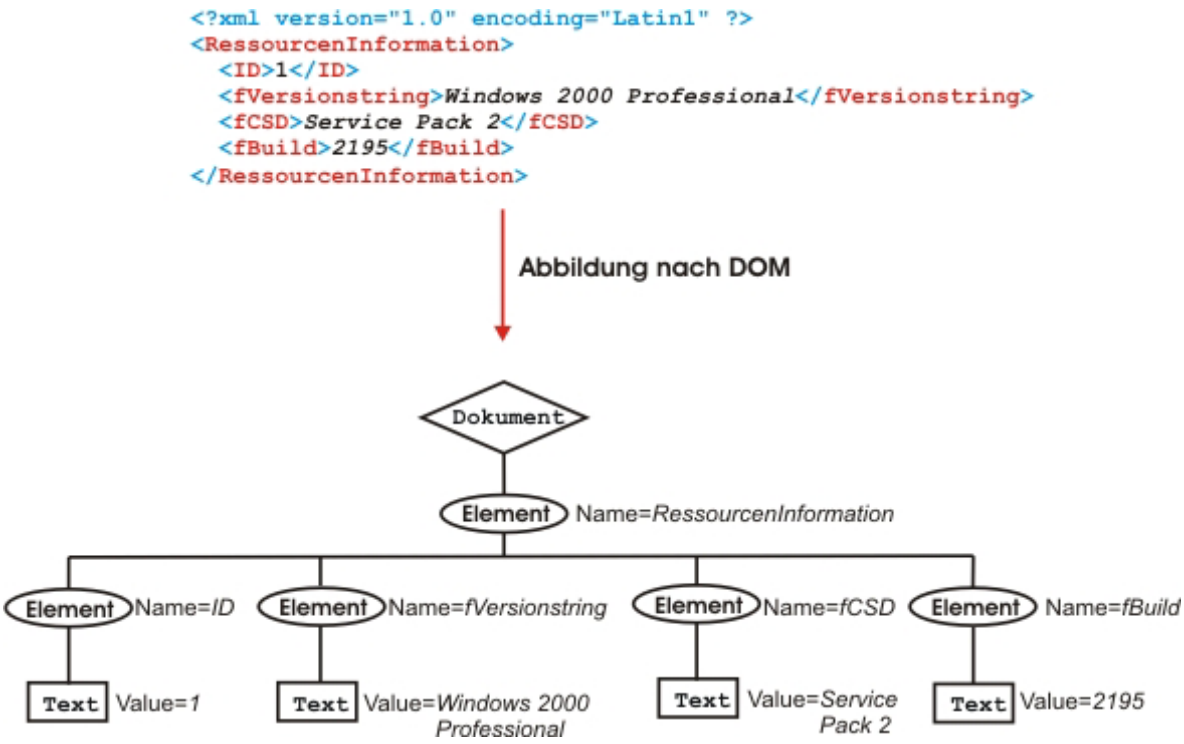


Abbildung 33 : Darstellung eines XML-Dokuments im oberen Bereich und deren Abbildung nach DOM im unteren Bereich

[1] Nähere Informationen zu DOM 1 und DOM 2 sind unter den angegebenen Quellen zu finden.

Um XML-Dokumente nach DOM zu konvertieren und umgekehrt, bedarf es eines Parsers. Beide Parser (XML->DOM und DOM->XML) sind im Paket *Open XML* enthalten.

4.2.2. Optimierung beim Speichern

Beim Speichern von Ressourcen-Informationen mehrerer Rechner in eine Datei (Grundgerüst dieser Datei siehe Abbildung 21) wären folgende Schritte notwendig, da jede Ressourcen-Information als XML-String zurückgeliefert wird und beim Einfügen in ein neues Dokument nach DOM konvertiert werden muss:

1. Anlegen eines neuen XML-Dokumentes als DOM
2. Für jeden Servereintrag:
    - Zurückliefern aller Ressourcen-Informationen als XML-String
    - Konvertieren des XML-Strings jeder Ressourcen-Information nach DOM und Importieren in das angelegte XML-Dokument
3. Konvertieren des DOM nach XML

Jedoch können diese Konvertierungen umgangen werden, indem die zu speichernde Datei aus Strings zusammengesetzt wird. Zuerst wird der Kopf der XML-Datei in den String geschrieben, danach für jeden Servereintrag der Kopf der Servereinträge, die Ressourcen-Informationen, die als XML-String zurückgeliefert werden, der Abschluss der Servereinträge und letztendlich der Abschluss der XML-Datei. Am Ende wurde so der String generiert, der auch beim finalen Konvertieren des XML-Dokumentes nach DOM entstanden wäre.
Um den Performancegewinn zu messen, wurden alle Ressourcen-Informationen einer bestimmten Anzahl von Servern abgefragt. Nachdem diese Abfrage beendet wurde, wurden diese Ressourcen-Informationen in eine XML-Datei gespeichert. Die Zeit, die zum Speichern der Ressourcen-Informationen (Exportieren mit Hilfe der Funktion *ExportXML* sowie das physikalische Speichern in eine XML-Datei) nach Beendigung der Abfrage notwendig war, wurde gemessen. In Abbildung 34 ist das Ergebnis dieser Messung dargestellt. Die x-Achse zeigt die Anzahl der gespeicherten Servereinträge und die y-Achse, die Zeit (in ms), die

zum Speichern notwendig war. Der blaue Kurvenverlauf charakterisiert die Speicherung mit Hilfe der Konvertierungen nach DOM bzw. XML, der gelbe Kurvenverlauf die Speicherung ohne Konvertierung. Aufgrund der Übersichtlichkeit ist die blaue Kurve nur bis zu einer Anzahl von zehn Servereinträgen dargestellt. Wie aus der Abbildung zu erkennen ist, besitzt die blaue Kurve einen exponentiellen Anstieg, wobei die gelbe Kurve einen linearen Verlauf annimmt. Bei einer Anzahl von vier Servereinträgen beträgt der Performancegewinn der Speichermethode ohne Konvertierung bereits das 7,6-fache gegenüber der Speichermethode, bei der Konvertierungen stattfinden, bei einer Anzahl von sechs Servereinträgen bereits das 12,5-fache und bei zehn das 20-fache. Somit kommt mit jedem Servereintrag zum bisherigen Performancegewinn eine Performancesteigerung um das 2-fache hinzu.

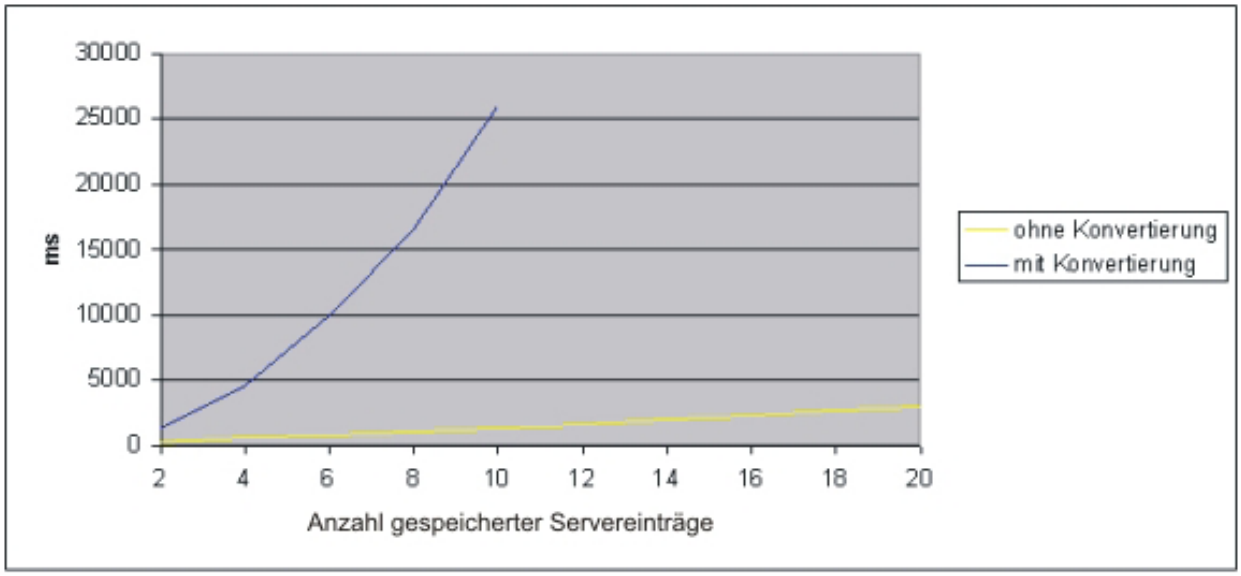


Abbildung 34 : graphische Darstellung der Performance bei der Speicherung von Ressourcen-Informationen abgefragter Serverprogramme in eine XML-Datei mit Hilfe von Konvertierungen und ohne

4.3. Format der Konfigurationsdaten

Die Daten der Konfigurationsdateien (ClientFile, ServerFile und ServerConfig-File) werden als zusammenhängender String gespeichert. Dabei werden die Daten durch ein Trennzeichen voneinander separiert. Dieses Trennzeichen ist *„ #1“* (Leerzeichen + #1). Die Methode, die diese Daten wieder ausliest, durchsucht den String, bis das Ende des Strings erreicht ist oder bis hinter dem Trennzeichen das Zeichen *#1* erneut auftritt, also *„ #1#1“*. Falls *„ #1#1“* auftritt, muss die Methode zum Auslesen der Daten erneut aufgerufen werden, um die restlichen Daten auszulesen. So können z.B. Einträge, die die gleichen Attribute besitzen, wie z.B. die Servereinträge des ServerFile, einfach in einer Schleife ausgelesen und in die entsprechenden Datentypen eingetragen werden. Abbildung 35 zeigt an einem Beispiel, wie Daten in einen String konvertiert und aus ihm wieder ausgelesen werden. Damit Attribute, die keine Daten enthalten, nicht irrtümlich als Stoppzeichen behandelt werden, wird als Trennzeichen nicht nur *#1* verwendet, sondern *#1* ein Leerzeichen vorangestellt.

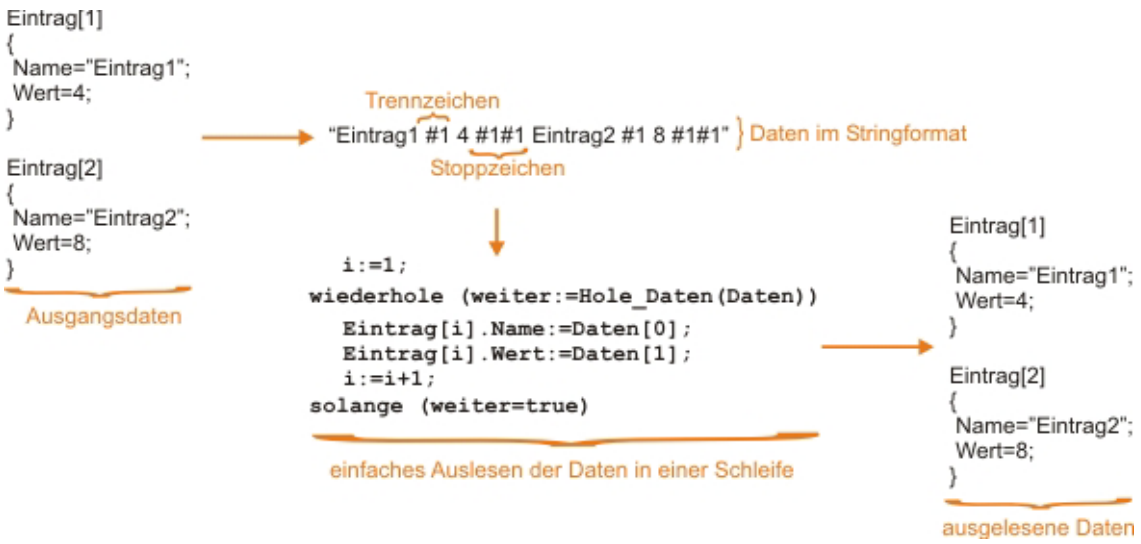


Abbildung 35 : Darstellung des Ablaufs der Konvertierung der Daten in einen String und des Auslesens aus einem String

4.4. Verschlüsselungsformat der Daten

Die Ressourcen-Informationen und Konfigurationsdaten im String-Format werden vor dem Übertragen bzw. Speichern verschlüsselt. Jeder verschlüsselte String besitzt eine Signatur bzw. einen Hashwert. Beide Daten werden ebenfalls in einem String gehalten und durch das Trennzeichen *#1* separiert. So können Daten und Signatur bzw. Hashwert in einem Schritt gesendet/empfangen bzw. gespeichert/geladen werden.

4.5. Übertragung der Daten

Für die Übertragung und den Empfang von Daten werden blockierende Methoden verwendet. Zwischen Server- und Clientprogramm findet bei der Kommunikation ein Wechselspiel zwischen dem Senden und dem Empfangen von Daten statt (vgl. Abbildung 26). Somit alternieren die Operationen *Senden von Daten* und *Warten auf Daten* bei beiden Kommunikationspartnern. Sobald eine Operation nicht ausgeführt wird, wird die Verbindung beendet. Mit Hilfe blockierender Methoden kann dieser Kommunikationsablauf einfacher realisiert werden, da die Abarbeitung in einer einzigen Methode erfolgen kann, da auf das Eintreffen der Daten gewartet werden kann. Damit Server- oder Clientprogramm nicht in undefinierte Zustände geraten, also bei einer blockierenden Methode stehen bleiben, arbeiten diese Methoden mit einem Timeout. Das bedeutet, dass die entsprechende Methode beendet wird, falls sie eine Operation nicht innerhalb einer bestimmten Zeit ausführen kann.

Da zur Übertragung TCP/IP verwendet wird, können nur maximal 8 KB an Daten in einem Paket untergebracht werden. Beim Senden größerer Datenmengen werden diese automatisch in mehrere Pakete aufgeteilt. Beim Empfangen muss jedoch der Empfangspuffer so oft ausgelesen werden, wie oft die Daten aufgeteilt wurden. So wird eine Größeninformation der zu sendenden Daten ebenfalls übermittelt, damit der Empfänger weiß, wie viele Daten er zu empfangen hat. Der Empfänger liest somit die Daten so lange aus dem Puffer, bis die übermittelte Größe erreicht ist. Da TCP bei Paketverlust automatisch das erneute Senden des Paketes anordnet und auch Duplikate und Reihenfol-

gevertauschung behandelt, ist garantiert, dass kein Paket verloren geht oder vertauscht wird. Es sei denn, die Verbindung ist gestört oder unterbrochen. Dann tritt jedoch der Timeout der blockierenden Methoden ein und die Verbindung wird beendet.

4.6. Ressourcen-Info-DLL

Die Methoden dieser DLL ermöglichen dem Nutzer, auf einfachste Art und Weise, die Ressourcen eines Rechners auszulesen. Die Methoden, die die DLL exportiert, können in zwei Bereiche unterteilt werden. Die Ressourcen-Informationen können auf zwei verschiedene Weisen ausgelesen werden. Zum einen kann ein Zeiger auf die Ressourcen-Information zurückgegeben werden, hinter dem die Informationen in einem *Record* (oder Struktur) gespeichert sind, also einfach ausgelesen werden können. Zum anderen können die Ressourcen-Informationen auch als String zurückgegeben werden, so dass sie in eine Datei gespeichert oder über eine Socket-Verbindung gesendet werden können.
Die Zeiger auf die entsprechenden Records der Ressourcen-Informationen, die zurückgeliefert werden, sind typisierte Zeiger, dass heißt, sie müssen nicht in einen entsprechenden Typ konvertiert werden, bevor sie dereferenziert werden können. Nachteil dabei ist, dass für jede Ressourcen-Information eine eigene Methode aufgerufen werden muss. Der Vorteil liegt darin, dass der Nutzer genau weiß, welchen Datentyp er auslesen muss, wobei Fehler durch falsche Konvertierung nicht auftreten können. Abbildung 36 verdeutlicht an einem Beispiel für Delphi den Unterschied zwischen typisierten und untypisierten Zeigern.
Im oberen Teil der Abbildung ist die Verwendung eines typisierten Zeigers und im unteren Teil die Verwendung eines untypisierten Zeigers dargestellt. Wie im unteren Teil zu erkennen ist, muss der untypisierte Zeiger vor der Dereferenzierung umgewandelt werden. Dazu ist es notwenig zu wissen, welcher Datentyp sich hinter dem Zeiger befindet.
Um dies dem Nutzer der Methoden der Ressourcen-Info-DLL zu ersparen, werden typisierte Zeiger und somit für jede Ressourcen-Information eine eigene Methode verwendet.

```
var X,Y:integer; // X,Y vom Typ integer
    P:^integer;  // Zeiger auf integer (typisierter Zeiger)
begin
  X:=8; // X einen Wert zuweisen
  P:=@X; // P die Adresse von X zuweisen
  Y:=P^; // P dereferenzieren; Ergebnis an Y zuweisen
end;

type
  Pinteger:^integer; // Pinteger ist Zeiger auf integer
var X,Y:integer; // X,Y vom Typ integer
    P:^Pointer;  // Allzweckzeiger (untypisierter Zeiger)
begin
  X:=8; // X einen Wert zuweisen
  P:=@X; // P die Adresse von X zuweisen
  Y:=Pinteger(P)^; // P in Zeiger auf integer konvertieren;
end;                  Ergebnis dereferenzieren; Ergebnis an Y zuweisen
```

Abbildung 36 : Verdeutlichung des Unterschieds zwischen typisierten und untypisierten Zeigern

4.6.1. Methoden

In diesem Abschnitt wird der allgemeine Umgang mit den exportierten Methoden der DLL erklärt.

Alle Methoden, die die DLL exportiert, sind Funktionen, die als Funktionsrückgabe-Wert ein Datum vom Typ *longword* (oder auch *DWORD*) zurückliefern. Dieser Rückgabewert entspricht einer Fehlerkonstante, deren Bedeutungen in der Quelltextdokumentierung nachgelesen werden kann. Alle Datentypen, die der DLL übergeben werden oder aus der DLL zurückgeliefert werden, sind Datentypen der Windows-API. Somit kann die DLL problemlos in Applikationen eingebunden werden, die in anderen Programmumgebungen als Delphi entwickelt wurden (wie z.B. C++ oder Visual Basic).

Vor Nutzung aller anderen Funktionen, muss der Nutzer zuerst die entsprechende Ressourcen-Informations-Klasse initialisieren. Dies geschieht mit Hilfe der Funktion *RS_Init*, die als Parameter die ID der Ressourcen-Informationen (vom Typ *longword*) erwartet. Welche Ressourcen-Information welche ID besitzt, kann ebenfalls der Quelltextdokumentation entnommen werden. Nach der Initialisierung hat der Nutzer die Möglichkeit, mit Hilfe der Funktion *RS_ExportXML*, die initialisierte Ressourcen-Information als XML-String zurückgeliefert zu bekommen. Diese Funktion erwartet ebenfalls die ID der Ressourcen-Information und eine Variable vom Typ *pchar*, dessen Zeiger auf die Ressourcen-Information im Stringformat gesetzt wird. Mit Hilfe der Funktion

RS_ImportXML hat der Nutzer die Möglichkeit, eine mit *RS_ExportXML* ausgelesene Ressourcen-Information in die entsprechende Ressourcen-Informations-Klasse zu importieren. Diese Funktion erwartet ebenfalls die ID der Ressourcen-Information, die importiert werden soll, sowie eine Variable vom Typ *pchar*, die auf die Daten zeigt, die importiert werden sollen.
Die Funktionen, mit denen ein typisierter Zeiger auf die Ressourcen-Informationen erlangt werden kann, sind in Tabelle 21 dargestellt.

Funktion	**Information über**
RS_Get_WinVersion	Windows-Version
RS_Get_Geraete	installierte Geräte
RS_Get_TCP	TCP/IP-Verbindungen
RS_Get_RoutTabelle	Routingtabelle
RS_Get_Laufwerke	Laufwerke
RS_Get_Speicher	Arbeitsspeicher und Swap-Datei
RS_Get_instSoftware	installierte Software
RS_Get_Prozessor	Prozessoren
RS_Get_NetFreigabe	Netzwerkfreigaben
RS_Get_MMGeraete	Multimedia-Geräte
RS_Get_Grafikkarte	primäre Grafikkarte
RS_Get_Aufloesungen	maximale Auflösungen
RS_Get_SMB_BIOSINFO	BIOS (allgemein)
RS_Get_SMB_SYSINFO	System
RS_Get_SMB_SYSENC	Chassis
RS_Get_SMB_BASEINFO	Base Boards, wie z.B. Motherboards
RS_Get_SMB_CPU	Prozessoren
RS_Get_SMB_MEMCTRL	Speichercontroller
RS_Get_SMB_MEMMOD	Speichermodule
RS_Get_SMB_CACHE	Prozessor-Cache
RS_Get_SMB_PORTCON	Ports
RS_Get_SMB_SLOTS	Slots
RS_Get_SMB_ONBOARD	Onboard-Geräte
RS_Get_SMB_LANG	unterstütze BIOS-Sprachen
RS_Get_SMB_PHYSMEM	physikalische Speicherfelder
RS_Get_SMB_MEMDEV	Speichergeräte der Speicherfelder

Tabelle 21 : Zusammenstellung der DLL-Funktionen zum Auslesen von Ressourcen-Informationen

Die genaue Beschreibung der einzelnen Ressourcen-Informationen, die ausgelesen werden können, ist unter Punkt 3.3 zu finden.
Welche Ressourcen-Information welchen Zeiger zurückliefert, kann ebenfalls der Quelltextdokumentation entnommen werden.
Wurden vor Aufruf einer der Funktionen aus Tabelle 21 Daten erfolgreich mit Hilfe der Funktion *RS_ImportXML* importiert, so werden diese Daten ausgelesen. Ansonsten werden die lokalen Ressourcen-Informationen zurückgeliefert.
Die initialisierten Ressourcen-Klassen müssen am Ende wieder freigegeben werden, damit der Speicher, den sie belegen, freigeben wird. Dazu dient die Funktion *RS_Free*, die die ID der freizugebenden Ressourcen-Information erwartet.
Abbildung 37 zeigt an Beispielen für Delphi, wie einfach die Methoden der DLL genutzt werden können, um Ressourcen-Informationen auszulesen, zu übertragen und anzuzeigen. Die erste Methode dieser Abbildung zeigt, wie Ressourcen-Informationen ausgelesen werden können, die z.B. über eine Socket-Verbindung übertragen werden sollen. Anhand der zweiten Methode wird gezeigt, wie z.B. über eine Socket-Verbindung empfangene Ressourcen-Informationen importiert und angezeigt werden können. Die dritte Methode ruft die entsprechende Methode zum Auslesen der entsprechenden Ressourcen-Informationen auf und gibt die Daten aus. Diese Methode, wie sie in der Abbildung dargestellt ist, kann nur einen Eintrag der Ressourcen-Information Windows-Version auslesen. Sie dient nur zur prinzipiellen Erläuterung der Benutzung der DLL-Funktionen. Die vierte Methode der Abbildung verdeutlicht, wie lokale Ressourcen-Informationen ausgelesen und angezeigt werden können.

ID der Ressourcen-Information

Beispiel einer Methode zum Versenden von Ressourcen-Informationen

```
procedure Senden(ID:longword);
var pData:pchar; // Zeiger auf Ressourcen-String
begin
  RS_Init(ID);// Ressource-Klasse initialisieren
  RS_ExportXML(pData,ID); // Ressource-Klasse als String zurückgeben
  Sende_Daten(pData); // Methode zum Versenden der Daten (nicht Bestandteil der DLL)
  RS_Free(ID); // Ressourcen-Klasse freigeben
end;
```

ID der Ressourcen-Information Zeiger auf Ressourcen-String

Beispiel einer Methode zum Empfangen von Ressourcen-Informationen

```
procedure Empfangen(ID:longword;pData:pchar);
begin
  RS_Init(ID); // Ressource-Klasse initialisieren
  RS_ImportXML(pData,ID); // Ressource-String importieren
  Anzeigen_Daten(ID); // Methode zum Anzeigen der Daten (siehe unten)
  RS_Free(ID); // Ressourcen-Klasse freigeben
end;
```

ID der Ressourcen-Information

Beispiel einer Methode zum Anzeigen von Ressourcen-Informationen

```
procedure Anzeigen_Daten(ID:longword);
var pWinVersion:PWin; // typisierter Zeiger auf Ressourcen-Informtions-Record
begin
  if (ID=Opt_WinVersion) then begin // Ressource-Information ist Windows-Version
    RS_Get_WinVersion(pWinVersion) // Windows_Version auslesen und Zeiger auf Information zurück
    write(pWinVersion^.Versionstring); // Zeiger dereferenzieren und Wert des Record-Feldes ausgeben
  end;
end;
```

ID der Ressourcen-Information

Beispiel einer Methode zum Anzeigen der lokalen Ressourcen-Informationen

```
procedure Res_Info(ID:longword);
begin
  RS_Init(ID); // Ressource-Klasse initialisieren
  Anzeigen_Daten(ID); // Methode zum Anzeigen der Daten (siehe oben)
  RS_Free(ID); // Ressourcen-Klasse freigeben
end;
```

Abbildung 37 : Darstellung der Nutzung der DLL-Funktionen am Beispiel

4.7. Das Serverprogramm

Um die Grundgerüste beider Versionen des Serverprogramms (als eigenständige Applikation bzw. Dienst) so klein wie möglich zu halten, wird der gesamte Kommunikationsablauf zwischen Server und Client in einer eigenen Klasse gekapselt – *TVerbindung*. Sobald das Clientprogramm mit dem Serverprogramm eine Verbindung aufgebaut hat, wird eine Instanz von *TVerbindung* erzeugt und die Abarbeitungsfunktion aufgerufen, welcher der Socket übergeben wird, über den die Kommunikation erfolgt. Somit ist es nicht notwendig, redundante Programmierung in beiden Serverprogramm-Versionen zu betreiben.
Zur Verwaltung der Server-Socket-Verbindungen wird die Komponente *TServerSocket* verwendet, welche in Delphi zu den Standardkomponenten gehört. Diese Komponente empfängt Anforderungen einer TCP/IP-Verbindung von anderen Rechnern und richtet die Verbindungen ein. Da bei der Kommunikation blockierende Methoden verwendet werden, ist es bei Verwendung dieser Komponente notwendig, einen eigenen Thread für die Kommunikation abzuspalten.
Weiterhin wird das Laden/Speichern und Ver-/Entschlüsseln der Konfigurationsdaten in der Klasse *TOptionen* gekapselt.

4.7.1. Serverprogramm als eigenständige Applikation

Damit ein Nutzer das Serverprogramm nicht ohne Berechtigung mit Hilfe des Taskmanagers beenden kann, ist es möglich, die Anwendung vor dem Taskmanager zu verbergen. Dazu wird die Anwendung mit Hilfe der Funktion *RegisterServiceProcess* der DLL *kernel32.dll* als Dienstanwendung registriert [Rek02]. Nach der Registrierung erscheint die Anwendung nicht mehr im Taskmanager. Unter Windows 2000/XP besteht jedoch die Möglichkeit, im Taskmanager alle laufenden Prozesse anzeigen zu lassen. In diesem Bereich ist das Serverprogramm jedoch sichtbar und kann beendet werden. Für den Betrieb unter Windows 2000/XP wurde unter anderem auch deswegen das Serverprogramm als Dienst implementiert.

Für das Anzeigen eines Icons in der Tray-Leiste von Windows wurde die Komponente *TCoolTray Icon* [Jak02] verwendet, welche das Registrieren und Abmelden das Iconeintrages übernimmt.

4.7.2. Serverprogramm als Dienst

Die Objektgalerie von Delphi bietet bereits das Grundgerüst für eine Service-Anwendung, die als Windows NT/2000/XP-Dienst läuft, an. Die Installation und Deinstallation des Serverprogramms erfolgt mit Hilfe des Dienstkontroll-Managers von Windows, indem das Serverprogramm mit den folgenden Startparametern ausgeführt wird:

- /INSTALL (installiert das Serverprogramm als Dienst)
- /UNINSTALL (entfernt das Serverprogramm aus den Diensten)
- /SILENT (Meldungen des Dienstkontroll-Managers unterdrücken)

Mit Hilfe des Programms Dienste der Microsoft Management Console kann das Serverprogramm gestartet, beendet und deinstalliert werden.

4.8. Das Clientprogramm

Die Ressourcen-Info-DLL verwaltet zur gleichen Zeit die Ressourcen-Informationen eines Rechners. Das Clientprogramm verwaltet jedoch die Ressourcen-Informationen mehrerer Rechner. Würde diese Funktionalität in die DLL integriert werden, würde sich die Größe der DLL mindestens verdoppeln. Das Serverprogramm und Applikationen, die die DLL verwenden, um nur die Ressourcen-Informationen eines Rechners auszulesen bzw. anzuzeigen, würden diese Funktionalität nicht benötigen. Die Verwaltung der Ressourcen-Informationen mehrerer Rechner muss außerhalb der DLL organisiert werden. Um den Programmieraufwand dabei so gering wie möglich zu halten und Redundanz zu vermeiden, ist es am besten, die Polymorphie-Methoden der Ressourcen-Informations-Klassen zu verwenden. Aus diesem Grund verwendet das Clientpro-

gramm die DLL nicht, sondern nutzt die folgenden Polymorphie-Methoden der Basisklasse:

- *Prozess* (Ressourcen-Information auslesen)
- *ExportXML* (Ressourcen-Information als XML-String zurückliefern)
- *ImportXML* (Ressourcen-Informations-XML-String importierten)
- *GetBaum* (baumartige Darstellung der Bezeichnung der Ressourcen-Information)
- *GetInfoAsArray* (Daten des gewählten Ressourcen-Informations-Eintrages in Tabellenform zurückliefern)

Die Klasse *TManage* des Clientprogramms kapselt diese Funktionalität. Neben der Verwaltung der Ressourcen-Informationen mehrerer Rechner kapselt *TManage* noch das Laden bzw. Speichern von Ressourcen-Abfragen sowie das Eintragen der Ressourcen-Informationen in die baumartige Darstellung (*Tree-view*) und in die tabellenartige Darstellung (*Stringgrid*) des Clientprogramms.
Wie auch im Serverprogramm, wird die Kommunikationsabwicklung des Client-programms mit dem Serverprogramm in der Klasse *TVerbindung* gekapselt.
Für die Verwaltung der Socket-Verbindung zum Serverprogramm wird die Komponente *TClientSocket* verwendet, die zu den Standardkomponenten von Delphi gehört.

4.9. Klassen des Abfrage-Systems

Das Ressourcen-Abfrage-System beinhaltet die in Tabelle 22 aufgelisteten Units und falls vorhanden, Klassen innerhalb der Units, die vom Clientprogramm, Serverprogramm bzw. der Ressourcen-Info-DLL verwendet werden.

Unit	Klasse	Funktion
Allgemein\Client	TManage	- Verwaltung der Ressourcen-Informationen mehrerer Server - Eintragen der Daten in Treeview und Stringgrid - Laden und Speichern von Ressourcen-Abfragen
Allgemein\DLLFunktionen		- Dynamisches Einbinden der Windows-DLLs (vgl. Punkt 1.2) - Deklaration der verwendeten Datenstrukturen der Windows-DLLs
Allgemein\Fehler		- Fehlerkonstanten-Definitionen der Funktionen der Ressourcen-Info-DLL
Allgemein\Kommunikation	TVerbindung	- Steuerung des Kommunikationsablaufes auf der Client- bzw. Serverseite
Allgemein\KommunikationDLL		- Dynamisches Einbinden der Ressourcen-Info-DLL
Allgemein\Krypt	TKrypt	- Steuerung des gesamten Ver- und Entschlüsselungsapparates
Allgemein\MStringgrid	TMyStringgrid	- Eigenes Stringgrid (abgeleitet von TStringgrid) mit Methoden zum Löschen und zur automatischen Größenanpassung
Allgemein\Optionen	TOptionen	- Laden/Speichern/Erstellen der Dateien ClientFile, ServerFile, ServerConfigFile
Allgemein\RegFunktionen		- enthält Methoden zum Finden und Auslesen von Informationen in bzw. aus der Registry
Allgemein\RessourcenInformationen		- enthält die IDs der RessourcenInformationen
Allgemein\RessourcenKlassen	TBasisKlasse Ressourcen-Informations-Klassen	- enthält TBasisKlasse und alle abgeleiteten Ressourcen-Informations-Klassen samt derer Methoden u.a. zum Auslesen, Exportieren nach XML, Importieren aus XML, Rückgabe als Baum und Tabelle der Ressourcen-Informationen
Allgemein\RessourcenTypen		- enthält Deklarationen der Typen und Datenstrukturen, die in den Ressourcen-Informations-Klassen verwendet werden
Allgemein\XMLFunktionen	TXML	- Schnittstelle zu DOM-Implementierung, DOM->XML- und XML->DOM-Parser

Tabelle 22 : Auflistung der Units samt Klassen des Ressourcen-Abfrage-Systems, die von Client-, Serverprogramm und Ressourcen-Info-DLL verwendet werden

Die in Tabelle 23 aufgelisteten Units werden nur im Clientprogramm verwendet und beinhalten Formulare zur Interaktion zwischen Programm und Nutzer.

Unit	Klasse	Funktion
Client\Main	Tfm_Main	- abgeleitet von TForm - enthält das Hauptfenster des Clientprogramms, sowie die Ereignis-Methoden der einzelnen Komponenten
Client\Unit_arbeite	Tfm_Arbeite	- abgeleitet von TForm - enthält des Status-Fenster, was anzeigt, wie weit eine Operation vorangeschritten ist
Client\Unit_Input	Tfm_Input	- abgeleitet von TForm - enthält das Passwort-Formular, das bei jeder Passwortabfrage erscheint
Client\Unit_Key	Tfm_Key	- abgeleitet von TForm - enthält das Formular, auf dem das ClientFile-Passwort geändert werden kann und ClientFiles erzeugt werden können
Client\Unit_Optionen	Tfm_Optionen	- abgeleitet von TForm - enthält das Formular, auf dem eingestellt werden kann, wann das Passwort abgefragt werden soll
Client\Unit_Server	Tfm_Server	- abgeleitet von TForm - enthält das Formular, auf dem die Einträge des ServerFile verändert werden können, sowie ServerConfigFiles angelegt werden können

Tabelle 23 : Auflistung der Units samt Klassen des Ressourcen-Abfrage-Systems, die nur vom Clientprogramm verwendet werden

Die Units, die nur durch das Serverprogramm als Dienst bzw. eigenständige Applikation verwendet werden, sind in Tabelle 24 aufgelistet.

Unit	Klasse	Funktion
Server95\Main	Tfm_Main	- abgeleitet von TForm - enthält das Hauptfenster des Serverprogramms als eigenständige Applikation, sowie die Ereignis-Methoden der einzelnen Komponenten
	TFileServerThread	- abgeleitete Klasse von TServerClientThread - Spaltet für jede Clientverbindung einen neuen Thread ab
Service2000\Unit_Main	TServerService	- abgeleitet von TService - Anmeldung/Abmeldung als Dienst von Windows
	TFileServerThread	- abgeleitete Klasse von TServerClientThread - Spaltet für jede Clientverbindung einen neuen Thread ab

Tabelle 24 : Auflistung der Units samt Klassen des Ressourcen-Abfrage-Systems, die nur vom Serverprogramm verwendet werden

Die exklusiv durch die Ressourcen-Info-DLL verwendeten Units sind in Tabelle 25 aufgelistet.

Unit	Funktion
DLL\ResInfo	- Unit der Ressourcen-Info-DLL
DLL\RessourcenFunktionen	- enthält alle Methoden, die die Ressourcen-Info-DLL exportiert (vgl. Punkt 4.6.1)

Tabelle 25 : Auflistung der Units samt Klassen des Ressourcen-Abfrage-Systems, die nur von der Ressourcen-Info-DLL verwendet werden

Eine umfangreichere Dokumentierung dieser Units und Klassen befindet sich in der Quelltextdokumentation der einzelnen Units.

4.10. Verwendete Komponenten

Die Implementierungen der Verschlüsselungsmethoden sind als Komponenten in die Klasse *TKrypt*, in der alle Verschlüsselungsmethoden gekapselt sind, eingebettet. Dies sind folgende Komponenten:

- Rijndael aus der Kollektion *DCPcrypt* [Bar02]
- RSA aus der Kollektion *Advanced Encryption Components* [TSM02]
- SHA-1 aus der Kollektion *Message Digests* [Sha02]

Ebenso verhält es sich mit den Implementierungen des DOM, des DOM->XML-Parsers und des XML->DOM-Parsers der Kollektion *Open XML* [Koe02], die in die Klasse *TXML* eingebettet sind.
Des Weiteren beruht das Auslesen des BIOS-Roms auf einer Unit, die von N. Bendlin [Ben01] geschrieben wurde, da unter Windows 2000 und XP nicht einfach über die Speicheradresse auf das BIOS-Rom zugegriffen werden kann. Es besteht nur die Möglichkeit, über das Kernel-Objekt *\Device\PhysicalMemory* oder mit Hilfe von Assembler-Befehlen das BIOS-Rom auszulesen.

5. Beispielabfragen

In diesem Kapitel werden Beispielabfragen anhand einiger Screenshots von Server- und Clientprogramm dargestellt.

Im ersten Beispiel werden alle Ressourcen-Informationen eines Rechners abgefragt und im Clientprogramm dargestellt. In Abbildung 38 ist ein Screenshot des Hauptfensters des Clientprogramms nach der Abfrage dargestellt. Auf der linken Seite des Hauptfensters wird in der Wurzel der Baumdarstellung die Server-ID, bestehend aus Rechnername, IP und Port des abfragten Servers, dargestellt. Diesem Eintrag sind die verfügbaren, erfolgreich abgefragten, Ressourcen-Informationen untergeordnet.

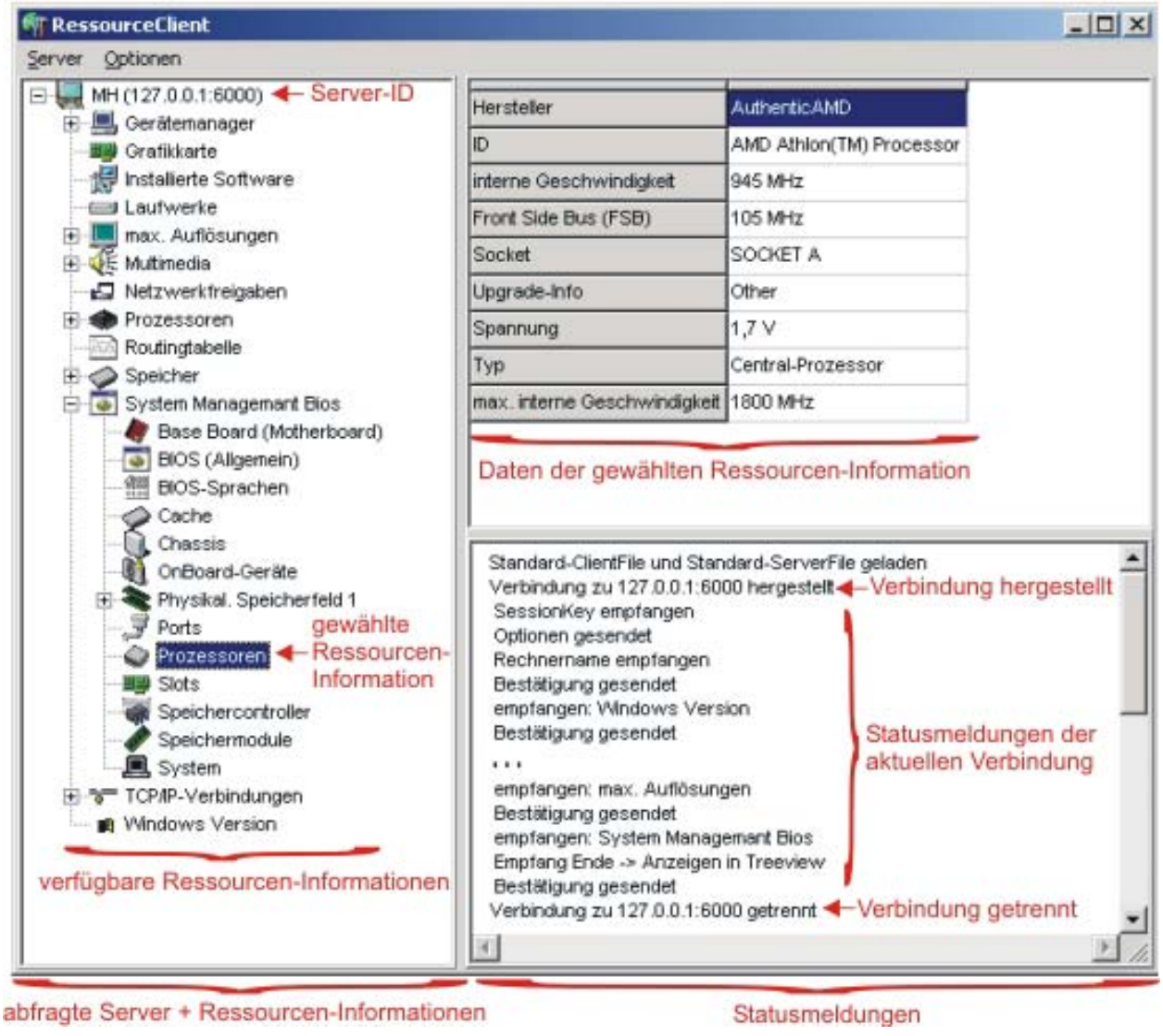


Abbildung 38 : Screenshot des Hauptfensters des Clientprogramms nach dem Ausführen der Beispielabfrage

Bei diesem Screenshot wurde der Eintrag *Prozessoren* der Ressourcen-Information *System Management BIOS* gewählt. Dessen Daten werden auf der rechten Seite im oberen Bereich des Hauptfensters dargestellt. Im unteren Bereich der rechten Seite des Hauptfensters werden alle Statusmeldungen ausgegeben. Dies sind unter anderem Verbindungsaufbau und -abbau sowie der aktuelle Status der bestehenden Verbindung.
Abbildung 39 zeigt einen Screenshot des Hauptfensters des Serverprogramms als eigenständige Anwendung nach der Abfrage. Wie auch das Clientprogramm, stellt das Serverprogramm den Kommunikationsverlauf, wie in der Abbildung dargestellt, in einem Statusfenster dar.

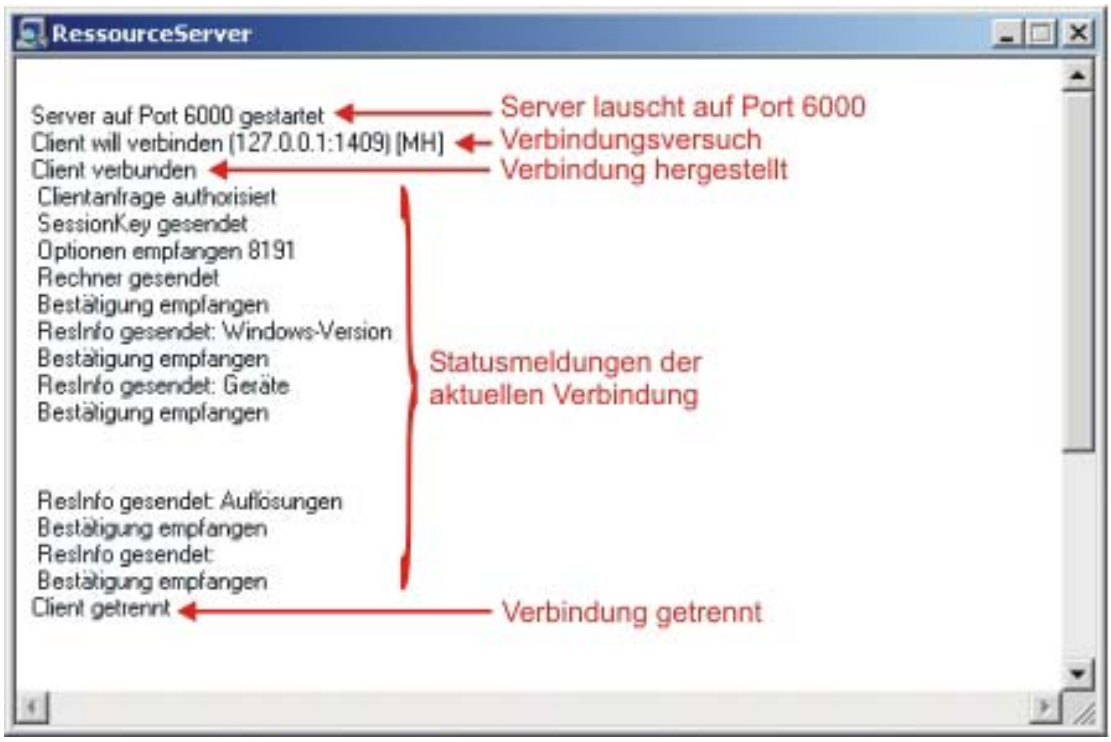


Abbildung 39 : Screenshot des Hauptfensters des Serverprogramms als eigenständige Anwendung nach dem Ausführen der Beispielabfrage

Basierend auf derselben Abfrage ist in Abbildung 40 die Ressourcen-Information *Gerätemanager* dargestellt. Wie die Abbildung zeigt, besitzt das Icon des Gerätes *Massenspeichercontroller* ein Ausrufezeichen. Dies bedeutet, dass es mit diesem Gerät ein Problem gibt. Aus den Eigenschaften *Gerätestatus* und *ConfigFlags* kann eine Beschreibung des Problems entnommen werden. Damit Geräte, mit denen es ein Problem gibt, sofort gefunden werden

können, wird dem Icon der Geräteklasse ebenfalls ein Ausrufezeichen hinzugefügt. Das Gerät *TEAC CD-540E*, dessen Eigenschaften ebenfalls in der Abbildung dargestellt sind, funktioniert fehlerfrei.

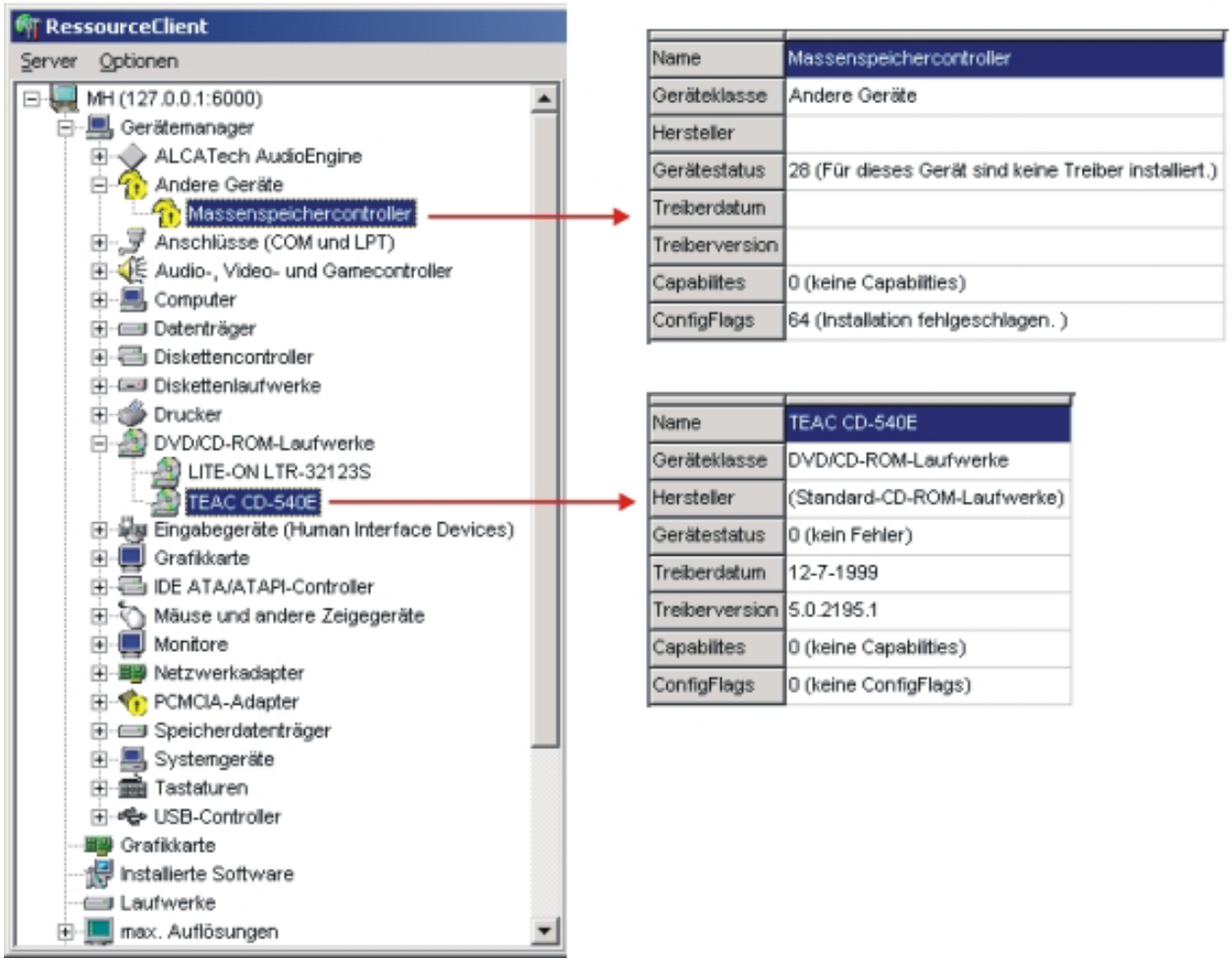


Abbildung 40 : Screenshot der Ressourcen-Information Gerätemanager (Hauptfenster Clientprogramm) nach dem Ausführen der Beispielabfrage

Dass auch das Abfragen und Darstellen mehrerer Server problemlos funktioniert, zeigt Abbildung 41, wo zehn Server abgefragt wurden und im Clientprogramm dargestellt sind.

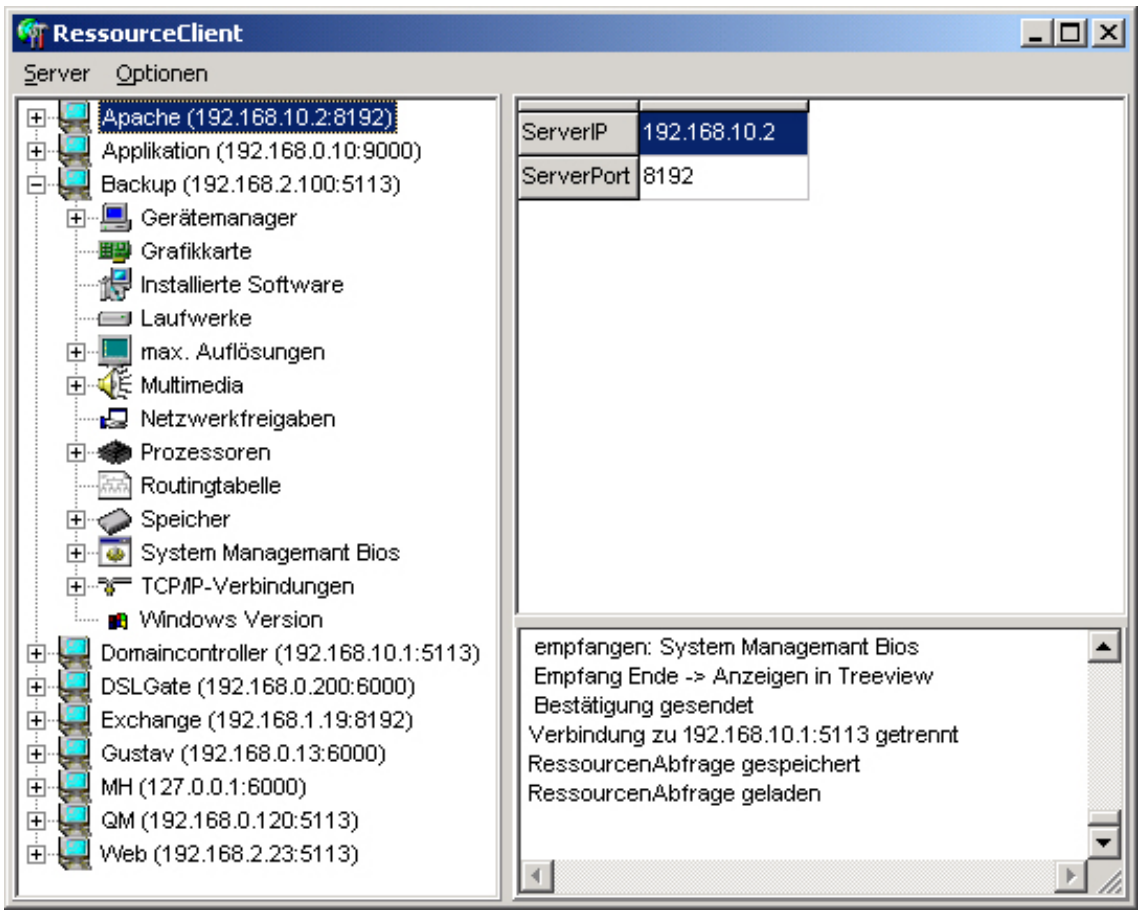


Abbildung 41 : Screenshot des Hauptfensters des Clientprogramms nach dem Abfragen von zehn Serverprogrammen

6. Lastmessungen

Die Lastmessungen wurden zwischen zwei direkt miteinander verbundenen Rechnern durchgeführt, wobei speziell über das Netzwerk keine Behinderungen bzw. Beeinflussungen durch andere Rechner stattfinden konnten.
Als Testrechner wurden Rechner verwendet, die heutzutage im mittleren Performancebereich liegen. Ihre Konfiguration ist in Tabelle 26 aufgelistet. Beide Rechner sind durch ein 100Mbit/s-Ethernet-Netzwerk miteinander verbunden.

Ressource	**Client-Rechner**	**Server-Rechner**
Prozessor	AMD Athlon Thunderbird 945 MHz	Intel Celeron 1300 MHz
Front Side Bus (FSB)	105 MHz	100 MHz
Arbeitsspeicher	256 MB 140 MHz	256 MB 100 MHz
Festplatte	IBM 40GB 5400 U/min UDMA 66	ST 40GB 5400 U/min UDMA 66
Netzwerkkarte	3Com Etherlink XL	Realtek 8139
Betriebssystem	Windows 2000 Professional SP2	Windows XP Home

Tabelle 26 : Auflistung der Ressourcen der verwendeten Rechner für die Lastmessungen

Zur Messung der prozentualen Auslastung der CPU und des Netzwerks[1] wurde der Taskmanager von Windows 2000 bzw. Windows XP verwendet. Um die genaue Datenmenge zu messen, die über das Netzwerk versendet wurde, wurde das Programm *SpyNet CaptureNet* [Nic00] verwendet, welches jedes Datenpaket erfasst und analysiert, das das Netzwerkinterface passiert.

[1] Der Taskmanager von Windows XP zeigt neben der prozentualen CPU-Belastung noch die prozentuale Auslastung des Netzwerkes an.

6.1. CPU-Auslastung

Bei dieser Lastmessung wurde die prozentuale Auslastung der CPU mit Hilfe des Taskmanagers von Windows 2000 bzw. XP erfasst. Die Aktualisierungsgeschwindigkeit wurde auf *Hoch* eingestellt, um einen detaillierteren Verlauf zu erhalten. Die CPU-Auslastung wird so aller 0,5 Sekunden aktualisiert. Die horizontale Achse der Darstellung im Taskmanager entspricht dem zeitlichen Verlauf der CPU-Auslastung, wobei der zeitliche Unterschied zwischen zwei vertikalen Linien drei Sekunden entspricht. Die vertikale Achse stellt die prozentuale CPU-Auslastung dar, wobei die Auslastung Werte zwischen 0% und 100% annimmt.

6.1.1. Clientprogramm

In der ersten Lastmessung wurde die CPU-Auslastung des Clientprogramms bei der Ressourcen-Abfrage an den Server gemessen. Dies beinhaltet folgende Operationen des Clientprogramms:

- Verbindungsaufbau zum Server
- Anlegen einer neuen Instanz der Verschlüsselungsklasse *TKrypt*
- Importieren des RSA-Schlüssels
- Autorisationsanfrage mit Hilfe von RSA verschlüsseln und versenden
- Sitzungsschlüssel empfangen und mittels RSA entschlüsseln
- Empfangen, Entschlüsseln mit Hilfe von Rijndael und Importieren der Ressourcen-Informationen des Servers
- Verbindungsabbau
- Eintragen der Ressourcen-Informationen in die Baumansicht (Treeview) des Clientprogramms

Die Länge des Sitzungsschlüssels umfasst immer eine maximale Länge von 256 Bit, die im Quelltext fest eingestellt ist. Da asymmetrische Verschlüsselungen rechenintensiver als symmetrische sind, wurden die Lastmessungen in Abhängigkeit von der Schlüssellänge des RSA-Schlüssels (512, 1024 bzw.

2048 Bit) durchgeführt. Dass die symmetrische Verschlüsselung die CPU wesentlich weniger belastet, wird später gezeigt. In Abbildung 42 ist die CPU-Auslastung des Clientrechners in Abhängigkeit von der Schlüssellänge des RSA-Schlüssels dargestellt, wobei vom Server alle verfügbaren Ressourcen-Informationen abgefragt wurden.

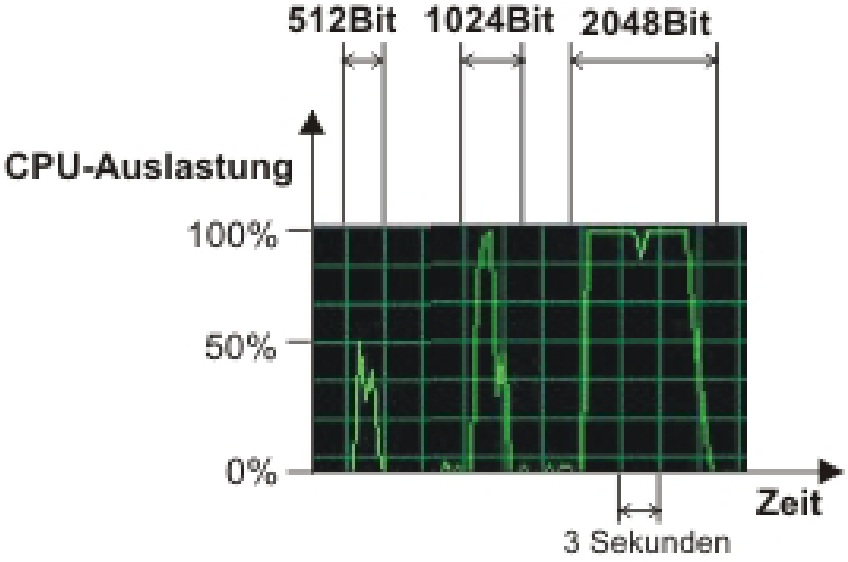


Abbildung 42 : Darstellung der CPU-Auslastung der Clientseite beim Abfragen aller Ressourcen-Informationen des Servers

Wie die Abbildung zeigt, wird die CPU bei einer Verwendung eines 512-Bit-RSA-Schlüssels nur ca. 3 Sekunden zu maximal 50 % belastet, bei einem 1024-Bit-RSA-Schlüssel ca. 4 Sekunden zu maximal 100 % und bei einem 2048-Bit-RSA-Schlüssel ca. 10 Sekunden, wobei hier die CPU fast durchgängig zu 100% belastet wird. Dies zeigt, dass die CPU-Belastung stark von der Länge des RSA-Schlüssels abhängt. Abbildung 43 verdeutlicht dies noch klarer, indem zum einem alle Ressourcen-Informationen (linker Graph der Abbildung) und zum anderen nur eine Ressourcen-Information (rechter Graph der Abbildung) vom Server, unter der Verwendung eines 2048-Bit-RSA-Schlüssels, abgefragt wurden. Des Weiteren zeigt die Abbildung, welche Operation wie lange und wie stark die CPU belasten. Anhand dieser Abbildung ist zu erkennen, wie groß der Anteil der RSA-Ver- bzw. -Entschlüsselung an der CPU-Auslastung ist. Das eigentliche Abfragen der Ressourcen-Information ist Operation fünf der Abbildung und nimmt bei dieser Schlüssellänge einen kleinen Anteil der CPU-Belas-

tung ein. Dies wird vor allem beim Vergleich beider Graphen deutlich. Die CPU-Belastung ist beim Abfragen einer bzw. aller Ressourcen-Abfragen fast gleich groß. Daran ist auch der Vorteil der hybriden Verschlüsselung in Punkto CPU-Belastung deutlich erkennbar (vgl. Punkt 2.3.2).

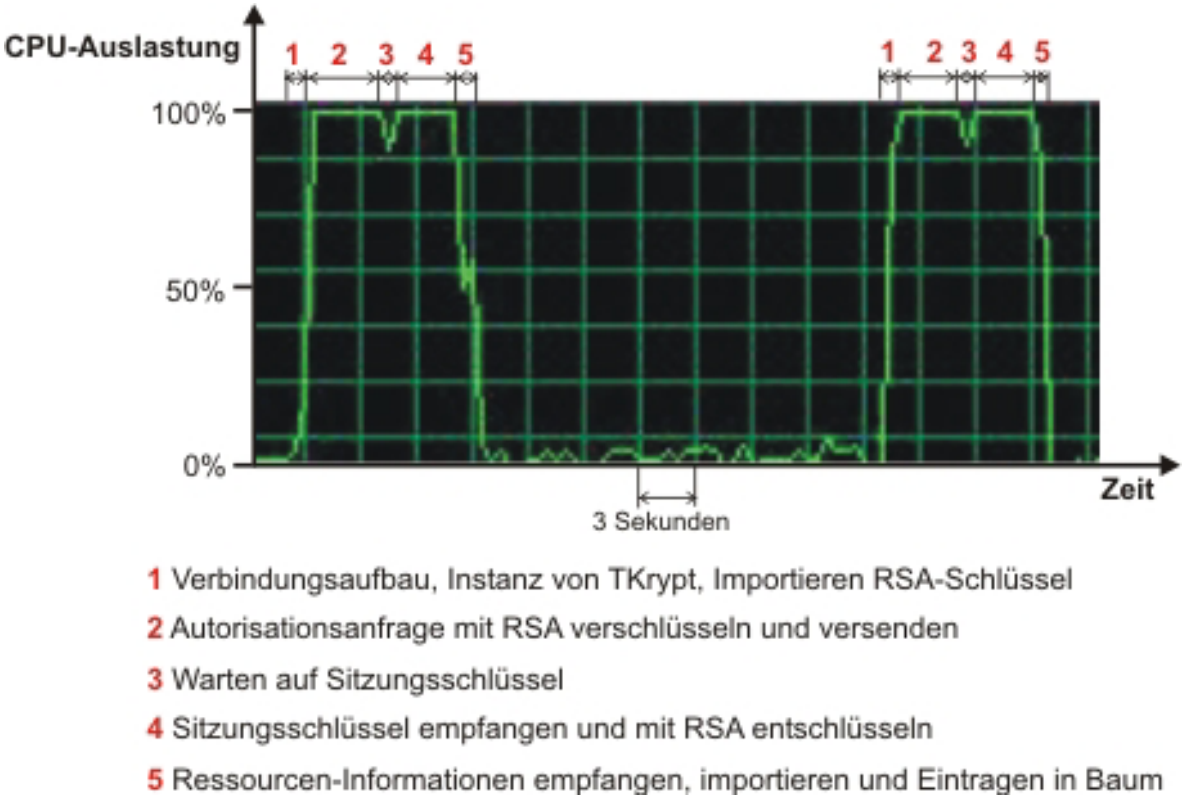


Abbildung 43 : Darstellung der CPU-Auslastung der Clientseite beim Abfragen aller (linker Graph) bzw. einer (rechter Graph) Ressourcen-Information(en) des Servers

6.1.2. Serverprogramm

Die CPU-Auslastung auf der Serverseite wurde ebenfalls bei der Ressourcen-Abfrage durch das Clientprogramm gemessen. Wie auch bei der Lastmessung auf der Clientseite wurde nach unterschiedlichen Schlüssellängen des RSA-Schlüssels (512, 1024 bzw. 2048 Bit) differenziert. Folgende Operationen werden auf der Serverseite bei einer Ressourcen-Abfrage durch das Client-programm durchgeführt:

- Anlegen einer neuen Instanz der Verschlüsselungsklasse *TKrypt*
- Importieren des RSA-Schlüssels
- Autorisationsanfrage mit Hilfe von RSA entschlüsseln und überprüfen
- Sitzungsschlüssel erzeugen, mit Hilfe von RSA verschlüsseln und zum Client senden
- Ressourcen-Informationen auslesen, exportieren, mit Hilfe von Rijndael verschlüsseln und zum Client senden

In Abbildung 44 ist die CPU-Auslastung auf der Serverseite beim Abfragen aller Ressourcen-Informationen durch das Clientprogramm dargestellt.

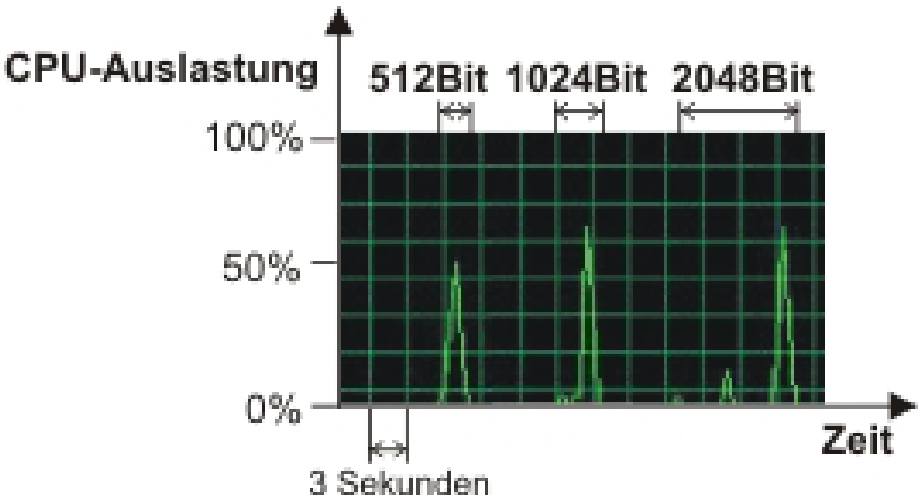
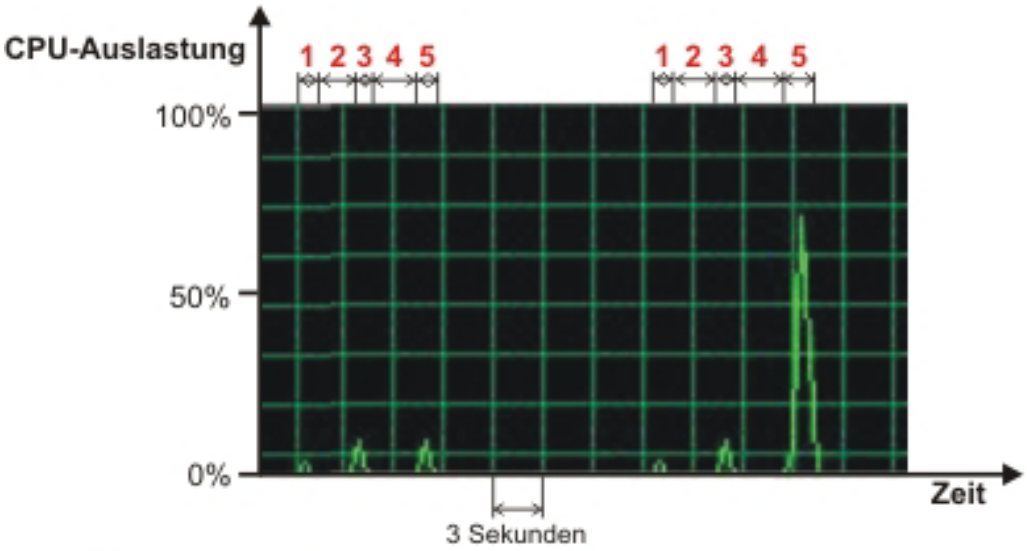


Abbildung 44 : Darstellung der CPU-Auslastung der Serverseite beim Abfragen aller Ressourcen-Informationen des Servers

Wie der Abbildung zu entnehmen ist, ist die CPU-Belastung bei der Verwendung von längeren RSA-Schlüsseln kaum höher, maximal 75%. Das liegt darin, dass als öffentliche RSA-Schlüssel meistens ein kurzer Schlüssel gewählt wird, dafür aber ein längerer privater RSA-Schlüssel. Somit wird das Ver- und Entschlüsseln von Daten mit Hilfe des öffentlichen Schlüssels durch weniger CPU-Auslastung und somit in kürzerer Zeit vollzogen als mit dem privaten Schlüssel. Das Serverprogramm belastet somit die CPU deutlich weniger als das Clientprogramm. Ein Anwender, der gleichzeitig an dem Rechner arbeitet, auf dem das Serverprogramm läuft, wird in seiner Arbeit damit nicht durch das Serverprogramm merklich behindert.

In Abbildung 45 wurden die CPU-Auslastungen beim Abfragen einer (linker Graph) bzw. aller (rechter Graph) Ressourcen-Informationen durch das Clientprogramm, bei einer Schlüssellänge des RSA-Schlüssels von 2048 Bit, gegenübergestellt. Im Gegensatz zur Clientseite, hat die Ver- und Entschlüsselung mit Hilfe von RSA auf die Serverseite eine geringere CPU-Belastung als das Auslesen, Exportieren, Verschlüsseln und Versenden der Ressourcen-Informationen. Jedoch dauert dies bei dem Testrechner nur ca. 2 Sekunden, bei einer maximalen CPU-Auslastung von 75%.



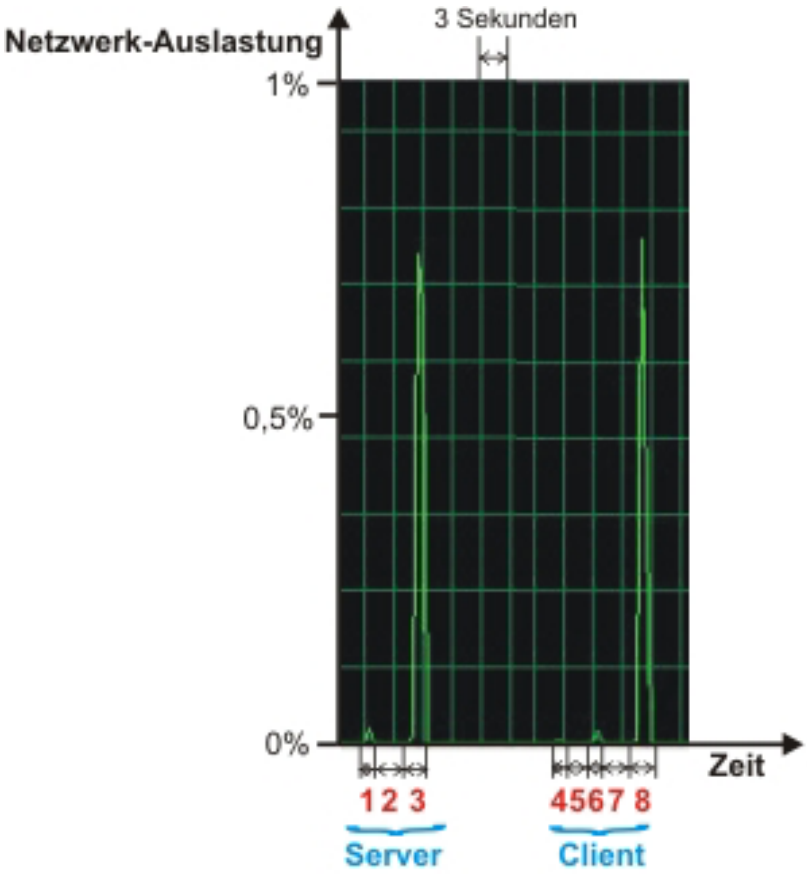
1 Instanz von TKrypt, Importieren RSA-Schlüssel
2 Warten auf Autorisationsanfrage
3 Autorisationsanfrage entschlüsseln mithilfe von RSA und überprüfen Sitzungsschlüssel generieren, mithilfe von RSA verschlüsseln und versenden
4 Warten auf Info, welche Ressourcen-Informationen gesendet werden sollen
5 Ressourcen-Informationen auslesen, exportieren, verschlüsseln mithilfe von Rijndael und versenden

Abbildung 45 : Darstellung der CPU-Auslastung der Serverseite beim Abfragen einer (linker Graph) bzw. aller (rechter Graph) Ressourcen-Information(en) durch das Clientprogramm

6.2. Netzwerk-Auslastung

Neben der CPU-Auslastung wurde auch die Auslastung des Netzwerkes durch das Ressourcen-Abfrage-System mit Hilfe des Taskmanagers von Windows untersucht, wobei die Darstellungsform der der CPU-Auslastung entspricht.

Da beide Rechner durch eine 100Mbit/s-Verbindung miteinander verbunden sind, entspricht eine maximale Auslastung von 100% 100Mbit/s.
Bei dieser Lastmessung wurden alle Ressourcen-Informationen des Serverrechners abgefragt. Abbildung 46 stellt die Netzwerkauslastung auf der Server- (linker Graph) bzw. auf der Clientseite (rechter Graph) dar. Des Weiteren sind die Operationen, die für die Netzwerklasten bzw. die Pausen zwischen diesen verantwortlich sind, in dieser Abbildung aufgelistet.



1 Empfang der Autorisationsanfrage und Versenden des Sitzungsschlüssels
2 Warten auf Info, welche Ressourcen-Informationen gesendet werden sollen
3 Versenden der Ressourcen-Informationen

4 Verbindungsaufbau
5 Erstellung der Autorisationsanfrage
6 Versenden der Autorisationsanfrage und Empfang des Sitzungsschlüssels
7 Entschlüsselung des Sitzungsschlüssels
8 Empfang der Ressourcen-Informationen

Abbildung 46 : Darstellung der Netzwerklast auf der Server- (linker Graph) bzw. Clientseite (rechter Graph) beim Abfragen aller Ressourcen-Informationen des Servers

Die Abbildung zeigt, dass eine maximale Auslastung von 0,8% des Netzwerkes erreicht wird. Somit beeinflusst das Ressourcen-Abfrage-System ein vergleichbares Netzwerk nicht merklich.

Wie viele Daten über das Netzwerk versendet werden, hängt zum einen von der Schlüssellänge und zum anderen von der Anzahl und der Größe der zu übertragenden Ressourcen-Information ab.

Tabelle 27 gibt einen Überblick über die Datenmenge, die bei einer Ressourcen-Abfrage in Durchschnitt vom Clientprogramm, Tabelle 28 vom Serverprogramm, übertragen werden. Dabei wird nach Schlüssellänge des RSA-Schlüssels, sowie der Anzahl der Ressourcen-Informationen (eine oder alle) unterschieden. Bei der Abfrage einer Ressourcen-Information wurde die Ressourcen-Information *Speicher* abgefragt, da die zu übertragende Datenmenge geringer als bei den anderen Ressourcen-Informationen ist, wobei die maximale Differenz bei der Datenübertragung gezeigt werden kann. Die Länge des symmetrischen Schlüssels beträgt wie bei den anderen Lastmessungen 256 Bit, da diese maximale Schlüssellänge fest im Quelltext gesetzt ist.

Schlüssel-länge	Authorisa-tionsanfrage	Bestätigung Rechnername	Welche RI[1]	Bestätigung Größen-Information	Bestätigung RI
2048 Bit	686 Byte	62 Byte	66 Byte	62 Byte pro RI	62 Byte pro RI
1024 Bit	344 Byte	62 Byte	66 Byte	62 Byte pro RI	62 Byte pro RI
512 Bit	174 Byte	62 Byte	66 Byte	62 Byte pro RI	62 Byte pro RI

Tabelle 27 : Überblick über die Datenmenge, die durchschnittlich vom Clientprogramm zum Serverprogramm versendet werden bei einer Ressourcen-Abfrage

[1] Ressourcen-Information (RI)

Schlüssel-länge	Sitzungs-schlüssel	Rechner-name	Größen-Information	Ressourcen-Information (RI)
2048 Bit	686 Byte	62-100 Byte	120 Byte pro RI	61458 Byte (alle) 346 (eine)
1024 Bit	344 Byte	62-100 Byte	120 Byte pro RI	61458 Byte (alle) 346 (eine)
512 Bit	174 Byte	62-100 Byte	120 Byte pro RI	61458 Byte (alle) 346 (eine)

Tabelle 28 : Überblick über die Datenmenge, die durchschnittlich vom Serverprogramm zum Clientprogramm versendet werden bei einer Ressourcen-Abfrage

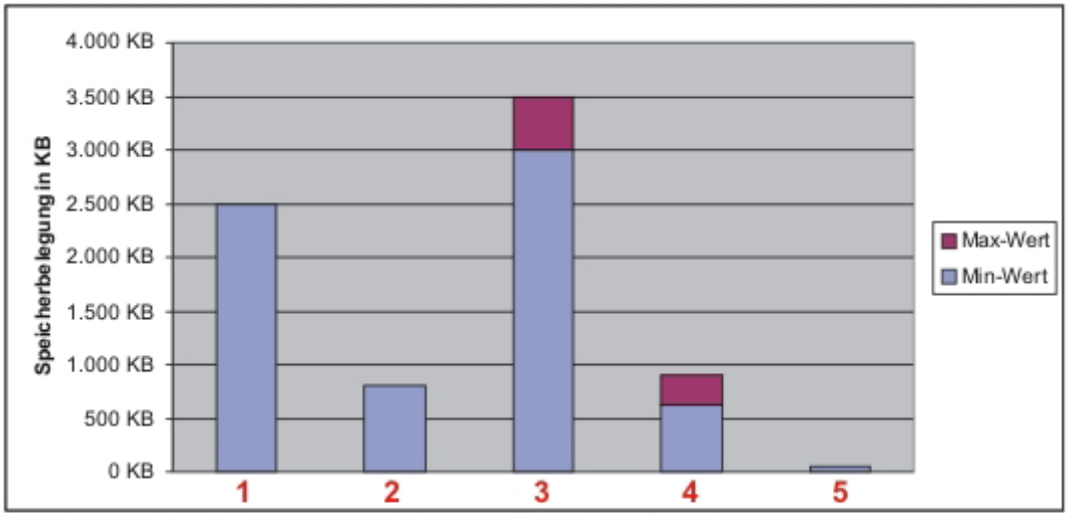
Wie den Tabellen entnommen werden kann, hat die Schlüssellänge des RSA-Schlüssels keinen so großen Einfluss auf die Menge des Datenverkehrs wie die Übertragung der Ressourcen-Informationen, dessen Anteil am Datenverkehr im Durchschnitt ca. 50% bis 97% des abgehenden Verkehrs des Serverprogramms beträgt.

6.3. Speicher-Belastung

Zwar zeigt der Taskmanager neben der CPU-Auslastung auch die Auslastung des Arbeitsspeichers in einem Kurvenverlauf an, jedoch sind in dieser Darstellung Schwankungen bis zu fünf MB nicht zu erkennen. Jedoch kann die Speicherbelegung jedes Prozesses bis in den KB-Bereich numerisch betrachtet werden. Die Analyse der Belegung des Arbeitsspeichers erfolgte über diese numerische Anzeige.

Das Clientprogramm belegt nach dem Start ca. 2,5 MB des Arbeitsspeichers, wobei das Serverprogramm ca. 800 KB belegt. Werden alle Ressourcen-Informationen des Serverrechners abgefragt, so belegt das Serverprogramm maximal 3 bis 3,5 MB des Arbeitsspeichers während dieser Abfrage. Das Clientprogramm belegt in diesem Fall ca. 620 bis 900 KB pro Serverabfrage. Wird eine gesamte Ressourcen-Abfrage in eine XML-Datei gespeichert, so benötigt jede Serverabfrage, wenn alle Ressourcen-Informationen abgefragt wurden, ca. 55 KB Speicherplatz. Somit kann eine XML-Datei, die bis zu 25 Serverabfragen enthält, bequem auf eine 1440KB-Diskette gespeichert werden, da diese 25 Abfragen ca. 1375 KB Speicherplatz (bei der Abfrage aller Ressourcen-Informationen) benötigen. Abbildung 47 zeigt einen graphischen Überblick über

diese Speicherbelegungen.



1 Speicherbelegung nach dem Starten des Clientprogrammes

2 Speicherbelegung nach dem Starten des Serverprogrammes

3 min./max. Speicherbelegung des Serverprogrammes beim Abfragen aller Ressourcen-Informationen durch das Clientprogramm

4 min./max. Speicherbelegung jeder Serverabfrage auf der Clientseite beim Abfragen aller Ressourcen-Informationen

5 Speicherbelegung jeder Serverabfrage, nach dem Speichern in eine XML-Datei, in der XML-Datei beim Abfragen aller Ressourcen-Informationen

Abbildung 47 : graphische Darstellung der Speicherbelegungen der aufgelisteten Zustände

6.4. Fazit

Anhand der Lastmessungen ist zu erkennen, welche Faktoren welche Bereiche am stärksten belasten bzw. beeinflussen. Somit ergeben sich die folgenden Schlüsse:

- Die Schlüssellänge des RSA-Schlüssels hat den stärksten Einfluss auf die CPU-Belastung, vor allem auf der Clientseite. Diese Belastung beträgt bei der Verwendung einer minimalen Schlüssellänge von 512 Bit ungefähr drei Sekunden, bei einer maximalen CPU-Belastung von 50%. Bei der Verwendung einer maximalen Schlüssellänge von 2048 Bit beträgt sie ungefähr neun Sekunden, bei einer maximalen CPU-Belastung von 100%.
- Die Anzahl und Größe der abgefragten Ressourcen-Informationen haben den stärksten Einfluss auf die Menge des Datenverkehrs zwischen Server- und Clientprogramm. Die zu übertragende Datenmenge liegt bei der reinen Übertragung der Ressourcen-Informationen zwischen ungefähr 470 Byte (eine Ressourcen-Information inklusive Größen-Information) und 63000 Byte (alle Ressourcen-Informationen inklusive Größen-Informationen).
- Die Anzahl der abgefragten Serverprogramme, sowie Anzahl und Größe der abgefragten Ressourcen-Informationen, haben den stärksten Einfluss auf die Größe des belegten Arbeitsspeichers auf der Clientseite, wobei pro Serverabfrage ungefähr 620 bis 900 KB, bei der Abfrage aller Ressourcen-Informationen, im Arbeitsspeicher belegt werden.

Damit zeigt sich, dass das Serverprogramm den Serverrechner kaum belastet, womit Arbeitsvorgänge an diesem Rechner nicht merklich behindert werden. Bei der Wahl der Schlüssellänge des RSA-Schlüssels sollte eine Mischung aus Performance- und Sicherheitsgewährleistung gefunden werden, damit das Clientprogramm den Clientrechner nicht überlastet. Sollte der Clientrechner den Anforderungen nicht genügen, besteht immer noch die Möglichkeit, das Abfragen der Ressourcen-Informationen aufzuteilen, so dass jedes Clientprogramm seinen abzufragenden Bereich hat. Da Ressourcen-Abfragen in eine

XML-Datei gespeichert werden, besteht hier die Möglichkeit, die XML-Dateien der verschiedenen Clientprogramme zu einer großen zusammenzuführen, wobei das gleiche Resultat erreicht wird, wie bei einer Ressourcen-Abfrage und Speicherung dieser Abfrage in einer XML-Datei eines einzelnen Clientprogramms.

7. Zusammenfassung und Ausblick

Kapitel eins gab einen Überblick, wo und in welcher Form Ressourcen-Informationen eines Rechners gespeichert werden. Dabei wurde detailliert auf die Registry von Windows eingegangen. Des Weiteren wurden Methoden bzw. Windows-DLLs erläutert, in denen diese Methoden, mit denen Ressourcen-Informationen ausgelesen werden können, die nicht in der Registry zu finden sind, gekapselt werden. Zusätzlich wurde beschrieben, auf welche Art und Weise Ressourcen-Informationen aus dem BIOS-Rom erfasst werden können. Somit erfährt der Leser, wo und wie detaillierte Ressourcen-Informationen eines Rechners ausgelesen werden können.

In Kapitel zwei wurden verschiedene Verschlüsselungstechniken und -methoden vorgestellt, die am häufigsten in der Praxis eingesetzt werden und als die sichersten gelten. Dabei wurde gezeigt, wie die verschiedenen Verschlüsselungstechniken miteinander kombiniert werden können, um Vor- und Nachteile der einzelnen Techniken zu minimieren bzw. zu eliminieren und somit ein optimales Maß an Sicherheit und Performance zu gewähren. Aufbau und Funktion der sichersten und am häufigsten eingesetzten Verschlüsselungsmethoden wurden ebenfalls in diesem Kapitel erläutert.

Die Konzeption des Ressourcen-Abfrage-Systems wurde in Kapitel drei diskutiert, wobei unter anderem auf Aufbau und Funktion von Server-, Clientprogramm und Ressourcen-Info-DLL, Kommunikationsprotokoll und der Kommunikationsverschlüsselung, Aufbau und Funktion der Konfigurationsdateien des Abfrage-Systems, Ablauf zwischen Client- und Serverprogramm sowie Bedienung der Applikationen eingegangen wurde. Des Weiteren enthält dieses Kapitel eine Auflistung aller Ressourcen-Informationen, die abgefragt werden können. In diesem Kapitel wurde gezeigt, wie das Konzept eines umfassenden Ressourcen-Abfrage-Systems aussehen kann. Dieses Konzept umfasst eine sichere Kommunikations- und Dateiverschlüsselung sowie einen Zugriffsschutz auf Server- und Clientprogramm. Des Weiteren werden Ressourcen-Informationen und Ressourcen-Abfragen im XML-Format gespeichert bzw. übertragen, so dass diese Daten in Systemen weiterverarbeitet werden können, die das Importieren von XML-Daten unterstützen oder direkt mit XML arbeiten. Das

Konzept des Ressourcen-Abfrage-Systems beinhaltet ebenfalls die Ressourcen-Info-DLL, in der die Methoden zum Abfragen der Ressourcen-Informationen gekapselt sind, womit es möglich ist, diese Methoden in anderen Systemen einzubinden.
Im anschließenden Kapitel wird die Implementierung dieses Systems erläutert. Im ersten Teil dieses Kapitels wurden die verwendeten Programmiertechniken und deren Vorteile beschrieben. Danach wurde erklärt, in welchen Formaten die Konfigurationsdateien des Systems und die Ressourcen-Informationen verschlüsselt, übertragen bzw. gespeichert werden. Anschließend wurde erläutert, welche Funktionen die Ressourcen-Info-DLL exportiert und wie diese Funktionen genutzt werden, um Ressourcen-Informationen auszulesen bzw. in einem speicher- bzw. übertragungsfähigen Format zu erhalten. Anhand dieses Kapitels lässt sich erkennen, wie problemlos die Ressourcen-Info-DLL zum Auslesen von Ressourcen-Informationen verwendet werden kann und wie einfach das System erweitert werden kann, da verschiedene Funktionalitäten in Klassen gekapselt sind.
Im Kapitel fünf wurden Beispielanfragen anhand von Screenshots gezeigt, wobei die dargestellten Informationen erläutert wurden.
Kapitel sechs zeigte an Lastmessungen, wie stark das Ressourcen-Abfrage-System CPU, Netzwerk und Arbeitsspeicher belastet. Anhand von detaillierten Abbildungen von Verlaufskurven der CPU- und Netzwerkauslastung wurde beschrieben, welche Operationen die CPU und das Netzwerk wie stark belasten. Dabei wurde die Auswirkung auf die Belastung durch verschiedene Schlüssellängen untersucht. Anschließend wurde in einem Fazit zusammengefasst, dass die Faktoren Schlüssellänge des RSA-Schlüssels die CPU, Anzahl und Größe der abgefragten Ressourcen-Informationen das Netzwerk und Anzahl der abgefragten Serverprogramme den Arbeitsspeicher am meisten belasten. Anhand dieser Lastmessungen wurde unter anderem gezeigt, dass das Serverprogramm die CPU des Rechners, auf dem es läuft, selbst bei der Verwendung der maximaler Schlüssellängen und der Abfrage aller Ressourcen-Informationen, nicht merklich belastet. Des Weiteren zeigen die Messungen der Netzwerk-Auslastung, dass das Ressourcen-Abfrage-System, bei der Übertragung aller Ressourcen-Informationen, ein 100 MBit/s-Netzwerk maximal 0,8% auslasten, wobei diese Belastung maximal eine Sekunde beträgt. Aufgrund dieser

Erkenntnisse wurden anschließend Empfehlungen gegeben, wobei bei der Wahl der Schlüssellänge geachtet werden sollte.
Kapitel sieben fasst die vorliegende Arbeit zusammen und beschreibt Erweiterungsmöglichkeiten des Ressourcen-Abfrage-Systems.
Die vorliegende Arbeit zeigt, dass es möglich ist, sehr detaillierte Informationen über einen Rechner zu erhalten, diese Informationen sicher zu verschlüsseln, ohne dabei die beteiligten Rechner, speziell die, auf denen das Serverprogramm läuft, stark zu belasten. Die Lastmessungen zeigen, dass das Serverprogramm während einer Ressourcen-Abfrage die Ressourcen des Rechners, trotz der Verwendung von Schlüssellängen von 2048 Bit (asymmetrischer Schlüssel) und 256 Bit (symmetrischer Schlüssel), nicht merklich belasten. Beim Online-Banking werden meist Schlüssellängen von 1024 Bit (asymmetrischer Schlüssel) und 128 Bit (symmetrischer Schlüssel) verwendet, was zeigt, wie hoch die Sicherheit der Kommunikation des Abfrage-Systems sein kann. Durch die Zusammenfassung von Funktionalitäten in Klassen, weist das System eine hohe Skalierbarkeit auf. Die verwendeten Verschlüsselungsmethoden könnten z.B. problemlos innerhalb einer Klasse ausgetauscht werden, ohne Änderungen an einer anderen Stelle vornehmen zu müssen. Des Weiteren ist es auch ohne großen Aufwand möglich, neue Ressourcen-Informations-Klassen dem System hinzuzufügen. Um das Auslesen der Ressourcen-Informationen in andere Anwendungen integrieren zu können, wurde die Ressourcen-Info-DLL entwickelt, dessen einfache Nutzung, trotz der aufwendigen internen Struktur, anhand von Beispielen in Kapitel vier (Implementierung) gezeigt wird. Außerdem zeigt das Abfrage-System, dass es unter Windows 95,98 und ME, bei denen es vom Betriebssystem keine Zugriffsbeschränkung auf Datei oder Schlüsselebene, in der Registry, gibt, die vom Nutzer verwendet werden kann, möglich ist, Sicherheitsaspekte umzusetzen.

7.1. Erweiterungsmöglichkeiten

Um einen Performancegewinn bei den Verschlüsselungsmethoden zu erreichen ist es sinnvoll, diese Methoden z.B. in C- oder C++-Code zu entwickeln, da diese Sprache schneller als Delphi arbeitet. Diese Methoden können dann in einer DLL gekapselt werden und in dem in Delphi entwickelten Abfrage-System eingebunden werden, ohne dass die Performance dieser Methoden beeinträchtigt wird. Somit belastet vor allem das Clientprogramm bei der Ver- und Entschlüsselung mit Hilfe von RSA die CPU nicht mehr so stark.
Um die CPU-Belastung weiter zu verringern ist es sinnvoll, asymmetrische Algorithmen zu untersuchen, die sich nicht auf Primzahlen, sondern auf elyptische Kurven stützen. Diese Algorithmen verwenden kürzere Schlüssellängen als RSA, womit sich auch die CPU-Belastung verringert. Da RSA, wie die Lastmessungen zeigen, den mit Abstand größten Anteil der CPU-Belastung verursacht, kann, wenn sich diese Algotihmen als einsetzbar erweisen, der größte Performancegewinn erreicht werden.
Eine weitere sinnvolle Erweiterung ist eine Implementierung einer direkten Kommunikation mit einer Datenbank - entweder direkt einer XML-Datenbank oder einer relationalen, die ein XML-Import-Modul besitzt. So könnten die Ressourcen-Abfragen im XML-Format direkt in eine Datenbank importiert werden.
Weiterhin könnte eine automatische Ressourcen-Abfrage implementiert werden, die z.B. über Kommandozeilenargumente gesteuert wird. Somit können, wenn die oben genannte Erweiterungsmöglichkeit implementiert wurde, die Ressourcen-Informationen der Rechner eines Netzwerks zu bestimmten Zeiten automatisch in der Datenbank aktualisiert werden. Dabei müsste vor allem bedacht werden, auf welche Art und Weise das Passwort für die Entschlüsselung des ClientFile übergeben bzw. gespeichert wird, ohne dabei die Sicherheit des Passwortes zu gefährden.

Abkürzungsverzeichnis

Abkürzung	Bedeutung
AES	Advanced Encryption Standard (siehe Seite 57)
AGP	Accelerated Graphics Port (Steckplatz für Grafikkarten, schneller als PCI)
API	Application Programming Interface (Schnittstelle für Anwendungsprogramme)
BIOS	Basic Input and Output System (Schnittstelle zwischen Hardware und Software)
bzw.	beziehungsweise
CA	Certificate Authority (Zertifizierungsstelle)
CD-ROM	Compact Disc – Read Only Memory (optisches Medium)
CLSID	Class Identification (Klassenindetifikation)
COM	Component Object Model (Standard für die Kompabilität von Komponenten)
CPU	Central Processing Unit (Zentrale Recheneinheit)
DAC	Discretionary Access Control (Zugriffsrechte eines Registry-Schlüssels)
DFÜ	Datenfernübertragung
DHCP	Dynamic Host Configuration Protocol (Protokoll für die automatischen IP-Vergabe)
DMA	Direct Memory Access (Direkter Speicherzugriff)
DOM	Document Object Model (siehe Seite 102)
d.h.	dass heißt
DES	Data Encryption Standard (siehe Seite 57)
DLL	Dynamic Link Library (dynamische Verweisdatei)
DNS	Domain Name Service (Protokoll für die Namensauflösung)
DWORD	Double Word (Datentyp)
ESDI	Extended Small Device Interface (veraltetes Interface zur Festplattenkontrolle)
GUID	Global Unique Identification (global eindeutige Indentifikation)
HKCC	HKEY_CLASSES_CONFIG
HKCR	HKEY_CLASSES_ROOT
HKCU	HKEY_CURRENT_CONFIG
HKDD	HKEY_DYN_DATA
HKEY	Hive Key (siehe Seite 8)
HKLM	HKEY_LOCAL_MACHINE
HKU	HKEY_USERS
HKPD	HKEY_PERFORMANCE_DATA
HTML	Hypertext Markup Language (siehe Seite 79)
ID	Identification (Identifikation)
IDE	Integrated Drive Electronics (Standard für Laufwerke)
INF	Dateien mit der Dateiendung *.inf*
INI	Dateien mit der Dateiendung *.ini*
IP	Internet Protocol (Internetprotokoll)
ISA	Industry Standard Architecture (alter Standard zur Datenübertragung zwischen einer Erweiterungskarte und der Hauptplatine)
I/O	Input/Output
KB	Kilobyte
longword	Entspricht DWORD

MAC (-Adresse)	Media Access Control (Hardwareadresse der Netzwerkkarte)
MB	Megabyte
Mbit/s	Megabit pro Sekunde
MD5	Message Digest Number 5 (siehe Seite 69,71)
ME	Millennium Edition (Windowsversion)
MHz	Megahertz
ms	Millisekunde
NIST	National Istitude of Standards and Technology (siehe Seite 58)
NSA	National Security Agency (siehe Seite 69)
NT	New Technology (Windowsversion)
NTFS	New Technology File System (erweitertes Dateisystem mit z.B. Zugriffsbeschränkung)
OLE	Object Linking and Embeding (Standard für Datenaustausch zwischen Anwendungen)
pchar	Pointer auf ein einzelnes Zeichen (Datentyp)
PCI	Peripheral Component Interconnect (interner Anschluß zu den Peripheriegeräten)
PCMCIA	Personal Computer Memory Card Interntional Association (Standards für kleine Erweiterungskarten für Notebooks)
PGP	Pretty Good Privacy (Email-Verschlüsselungsprogramm)
POST	Poer On Self Test (siehe Seite 50)
PS/2	Schnittstelle für Ein-/Ausgabegeräte wie Maus und Tastatur
RAM	Random Access Memory (Speicherbereich in den geschrieben werden kann)
RAMDAC	Random Access Memory Digital-to-Analog Converter (Mikrochip, der digitale Bilddaten in analoge Daten für die Anzeige umwandelt)
RI	Ressourcen-Information
ROM	Read Only Memory (Speicher, der schreibgeschützt ist)
RSA	Rivest-Shamir-Adleman (siehe Seite 64)
SAM	Security Accounts Manager (siehe Seite 31)
SCSI	Small Computer System Interface (Schnittstelle für Peripheriegeräte)
SGML	Standard Generalized Markup Language (siehe Seite 79)
SHA-1	Secure Hash Algorithm Version 1 (siehe Seite 69)
SID	Security Identification (Sicherheitsidentifkation)
SMBIOS	System Management BIOS
SSL	Secure Socket Layer (Verschlüsselungsprotokoll)
Task-Leiste	Bereich des Windows-Desktop, in dem schnell auf gestartete Programme zugegriffen werden kann
Tray-Leiste	Bereich des Windows-Desktop, in dem Programme einen Iconeintrag platzieren können
TCP	Transmission Control Protocol (verbindungsorientiertes Übertragungsprotokoll)
UDP	User Datagram Protocol (verbindungsloses Übertragungsprotokoll)
u.a.	unter anderem
USB	Universal Seriel Bus (Interface zu Peripheriegeräten)
vgl.	vergleiche
x86	Rechnerarchitektur, zu der 386, 486 etc. gehören
XOR	Exklusives Oder
XML	Extensible Markup Language (siehe Seite 79)
XP	Experience (Windowsversion)
z.B.	zum Beispiel

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[*] Im Rahmen dieser Arbeit durch den Autor herausgefundene Informationen über den Inhalt der Registry

[Bar02] D. Barton: *DCPcrypt*
http://www.scramdisk.clara.net/
gelesen April 2002

[Ben01] N. Bendlin: *BIOS Help*
http://www.swissdelphicenter.ch/de/showcode.php?id=748
SwissDelphiCenter.ch, 2001

[Ben02] D. Benderman: *SSL-Dokumentation*
http://atiswww.ira.uka.de/itdienste/ssl/
Universität Karlsruhe, 2002

[Bor02] Borland: *Delphi*
http://www.borland.com/delphi/index.html, 2002

[Bor98] G. Born: *Arbeiten mit der Registrierung von Windows 98*
Microsoft Press, Unterschleißheim, 1998

[Box01] D. Box, J. Lam, A. Skonnard: *Essential XML*
Addison-Wesley, München, 2001

[Bra00] D. Bratko, J. Farmer, P. Lipp, W. Platzer, A. Sterbenz: *Sicherheit und Kryptographie in Java*
Addison-Wesley, Bonn, 2000

[Bur01] S. Burnet, S. Paine: *Kryptographie*
mitp-Verlag GmbH, Bonn, 2001

[Com02] Computer Hope: *Windows Device Manager*
http://www.computerhope.com/win95er.htm, 2002

[Dae99] J. Daemon, V. Rijmen: *The Rijndael Block Cipher*
http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf, 1999

[Dis02] Distributed Management Task Force (DMTF): *System Managment BIOS (SMBIOS) Reference Specification Version 2.3.3*
http://www.dmtf.org/standards/bios.php, Mai 2002

[Dob00] W. Doberenz, T. Kowalski: *Borland Delphi 5 – Grundlagen und Profiwissen*
Hanser Verlag, Wien, 2000

[Gol01] Dr. C. F. Goldfarb: *Current Text of ISO 8879 (SGML)*
http://www.sgmlsource.com/8879/index.htm, 2001

[Hae02] J. Hähnle: Fehlercodes *im Gerätemanager und ihre Bedeutung*
http://www.paules-pc-forum.de/fehlermeld/fehler06.htm, 2002

[Hor02] R. Horn: *Windows-Tuning*
http://freenet.meome.de/app/fn/artcont_portal_news_article.jsp/73179.html
gelesen Mai 2002

[Hun01] T. Hunger: Diplomarbeit: *Kryptographie – Der Rijndael (AES) Algorithmus: eine Visualisierung*
http://home.datacomm.ch/th.aes/Daten/
Fachhochschule beider Basel Nordwestschweiz, 2001

[Jak02] T. Jakobson: *TCoolTray Icon*
www.delphi32.com
gelesen Mai 2002

[Knu81] D. Knuth: *The Art of Computer Programmings: Volume 2, Seminumerical Algorithms*
2nd Edition, Addison-Wesley, Bonn, 1981

[Koe02] D. Köhler: *Open XML*
http://www.philo.de/xml/, 2002

[Kof01] R. Koffmane: *BIOS*
SYBEX-Verlag, Düsseldorf, 2001

[Kos00] A. Kosch: *Delphi Win32 Lösungen – Das Zusammenspiel von Delphi mit dem Betriebssystem*
Software & Support Verlag GmbH, Frankfurt am Main, 2000

[Mac97] I. Macherius: *XML: Professionelle Alternative zu HTML*
http://www.heise.de/ix/artikel/1997/06/106/
IX, Juni 1997

[Mic02] Microsoft: *Windows Driver Development Kits (DDK)*
http://www.microsoft.com/ddk, 2002

[Mic02a] Microsoft: *Developer Network (MSDN)*
http://msdn.mircosoft.com, 2002

[Mic98] Microsoft TechNet: *Chapter 31 – Windows 98 Registry*
http://www.microsoft.com/technet/default.asp, 1998

[Nic00] L. Nicula: Spynet *CaptureNet 3.12 Build 4*
http://www.programfiles.com/default.asp?LinkID=7715, 2000

[Pen01] J. C. Penman: *IP Helper API*
http://www.netti.hu/doc/delphi_zine/IP%20Helper%20API%20Part%20I.htm
Delphi Informant Magazine, 2001

[Rob00] P. Robichaux: *Managing the Windows 2000 Registry*
O'Reilly, Sebastopol(California), 2000

[Rek02] F. Recktenwald: *Tipps & Tricks*
http://www.delphimeister.de
gelesen August 2002

[RSA99] RSA Laboratories: *Factorization of RSA-155*
http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html
RSA Laboratories, 1999

[Sch96] B. Schneier: *Angewandte Kryptographie*
Addison-Wesley, Bonn, 1996

[Sha02] D. Shapiro: *Message Digests*
http://www.csd.net/~daves/delphi/index.html
gelesen April 2002

[Sil01] R. Silverman: *A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths*
http://www.rsasecurity.com/rsalabs/bulletins/bulletin13.html
RSA Laboratories, 2001

[Sys02] Sysinternals: *Regmon for Windows*
http://www.sysinternals.com/ntw2k/source/regmon.shtml, 2002

[Tri02] Triplebyte: *System Device Manager Error Codes*
http://www.triplebyte.com/help/windows/errorcodes.htm
gelesen Juni 2002

[TSM02] TSM Inc.: *Advanced Encryption Components*
http://crypto-central.com/
gelesen April 2002

[Ulb96] A. Ulbrich: *BIOS – Aufbau und Funktion*
http://www.tu-chemnitz.de/informatik/RA/kompendium/
TU Chemnitz, 1996

[Wal01] F. Walther: *Registry Guide*
Markt+Technik Verlag, München, 2001

[Wob01] R. Wobst: *Abenteuer Kryptologie*
3. Auflage, Addison-Wesley, Bonn, 2001

[W3C98] W3C (World Wide Web Consortium) XML Working Group:
Extensible Markup Language (XML) 1.0
http://www.w3c.org/XML, Februar 1998

[W3C98a] W3C: *Document Object Model (DOM) Level 1 Specification*
http://www.w3.org/TR/REC-DOM-Level-1/, Oktober 98

[W3C99] W3C: *HTML 4.01 Specification*
http://www.w3.org/TR/html4/, Dezember 1999

[W3C00] W3C: *Document Object Model (DOM) Level 2 Core Specification*
http://www.w3.org/TR/DOM-Level-2-Core/, November 2000

[Wor99] J. Woram: *Windows 98 Registry*
Franzis-Verlag GmbH, Poing, 1999

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

.....
Matthias Hönemann

Leipzig, den 1. Oktober 2002