



Studienarbeit

Leitungsverschlüsselung

Student: Thorsten Ferres

Matrikelnr.: 159442

Kurs: TIT97ANM

Fachrichtung: Informationstechnik

Vertiefungsrichtung: Netzwerk- und Medientechnik

Betreuer: Prof. J. Schmidt

Ehrenwörtliche Erklärung:

Hiermit versichere ich ehrenwörtlich, die Studienarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt zu haben.

Unterschrift - Thorsten Ferres

Abstract

Diese Studienarbeit beschreibt den Aufbau und die Programmierung eines Leitungsverchlüssellers.

Zunächst werden der Aufbau und die Arbeitsweise des eigens für dieses Projekt entwickelten Verschlüsselungsalgorithmus FC1 aufgezeigt.

Anschließend wird auch insbesondere auf die Anforderungen an das zu verwendende Protokoll FCP eingegangen, welches ebenfalls selbst erdacht und implementiert wurde.

Im Anschluß daran folgt eine Beschreibung der Implementation der beiden ersten Teilen in einem Mikrocontroller-Programm.

Schließlich wird das ebenfalls für dieses Projekt entwickelte Programm ‚Datenterminal‘ beschrieben, welches auf einem PC unter Windows 9x ausführbar ist und mit dem Leitungsverchlüsseler unter Einhaltung des entwickelten FCP-Protokolles kommuniziert sowie zur Übertragung von Schlüsselinformationen an dieses Gerät dient. Es stellt dem Benutzer aber auch noch eine Reihe von weiteren Funktionen zur Verfügung, z. B. das lokale Verschlüsseln der Daten.

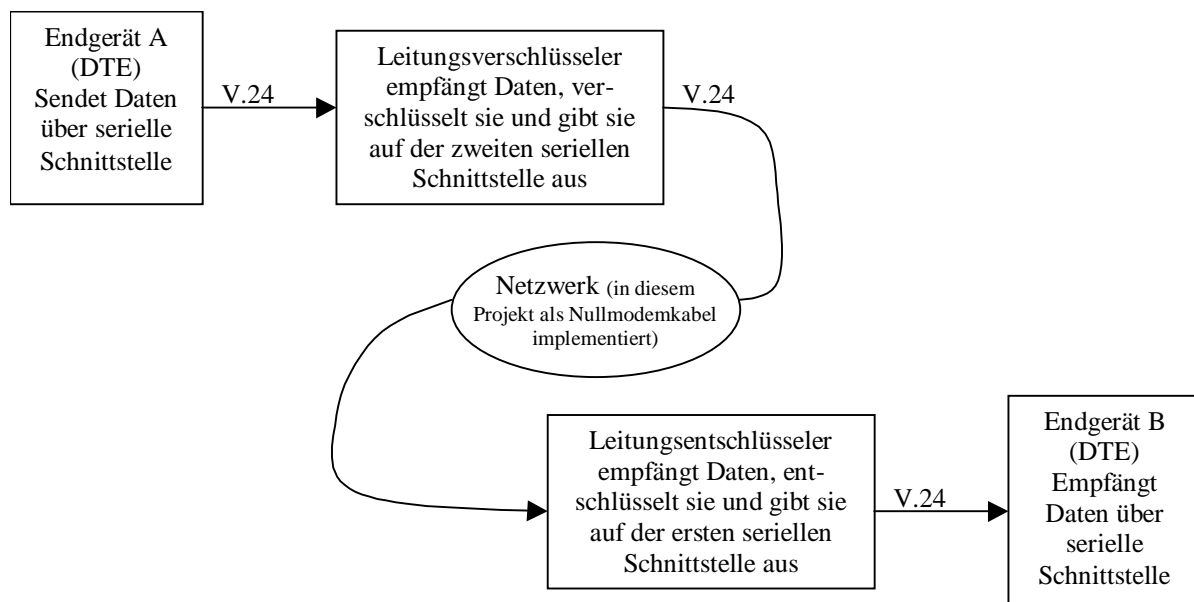


Abb. 1 - Das Schema der Leitungsverchlüsselung

Inhalt

Studienarbeit Leitungsverchlüsselung	1
Abstract	3
1. Aufgabenstellung	6
1.1. Notwendige Elemente zum Bau eines Leitungsverchlüssellers	6
1.2. Lösungskonzept	6
2. Der Verschlüsselungsalgorithmus FC1	8
2.1. Die Funktionsweise des Verschlüsselungsalgorithmus	8
2.2. Die Entschlüsselung	9
2.3. Stärken und Schwächen dieses Verschlüsselungsverfahrens	9
2.4. Ein Brute-Force-Angriff auf den FC1-Algorithmus	11
3. Das Kommunikationsprotokoll FCP	14
3.1. Die einzelnen Schritte der Übertragung	14
3.2. Die Prüfsumme und das Acknowledge	16
3.3. Bewertung des Protokolles	17
3.4. Der Gesamt Ablauf des Protokolls	18
4. Verwendete Werkzeuge	19
4.1. Die eingesetzte Hardware	19
4.2. Zur Entwicklung eingesetzte Software	20
5. Das Mikrocontroller-Programm	21
5.1. Das Hauptprogramm	21
5.2. Die Unterprogramme zur seriellen Übertragung	22
5.3. Die Routinen zur Ver- bzw. Entschlüsselung der Daten	24
5.4. Der logische Ablauf des Mikrocontroller-Programmes	25
6. Das Windows-Programm ‚Datenterminal‘	26
6.1. Aufbau des Programmes ‚Datenterminal‘	26
6.1.1. Die Anwendungsklasse ‚CHexView‘ und die Klasse ‚CMainFrm‘ zur Erstellung des Hauptfensters	26
6.1.2. Die Klasse ‚CHexViewDoc‘ zur Erstellung eines Dokumentes	26
6.1.3. Die Klasse ‚CHexViewView‘ zur Darstellung und Manipulation der Daten	27
6.1.4. Die Klasse ‚CSerial‘ zur Kommunikation über die serielle Schnittstelle	33
6.1.5. Dialogklassen zur Benutzerkommunikation	34
6.1.6. Die Klasse ‚KeyManagement‘ zur Verwaltung der Schlüssel	34
6.1.7. Das Klassenmodell	36
6.2. Fehler oder Fehlfunktionen im Programm ‚Datenterminal‘	36
6.2.1. Fehler im Anzeigemodus	36
6.2.2. Fehler im Editiermodus	37
7. Diskussion	38
Anhang	39
A - Die Bedienung des Programmes ‚Datenterminal‘	39
a) Das Menü ‚Datei‘	39
b) Das Menü ‚Bearbeiten‘	40

c) Das Menü ‚Ansicht‘	40
d) Das Menü ‚Senden‘	40
e) Das Menü ‚Verschlüsselung‘	41
f) Das Menü Hilfe	42
B – Die Bedienung des Leitungsverschlüssellers	42
a) Die Verbindungskabel	42
b) Stromversorgung	43
c) Betrieb des Leitungsverschlüssellers	44
Abbildungsverzeichnis	45
Glossar	46
Index	49
Quellenverzeichnis	51

1. Aufgabenstellung

1.1. Notwendige Elemente zum Bau eines Leitungsverschlüsslers

Seit der Erschließung des Internet für den Consumer-Markt spielt die Datenübertragung im Leben der Menschen eine immer größere Rolle. Und dabei wird es in Zukunft auch immer mehr darum gehen, inwiefern sich auch sensible Daten über die neuen Medien sicher übertragen lassen. Sicher heißt in diesem Zusammenhang, dass der Inhalt der Daten weder von Dritten abgehört noch verfälscht werden kann. Sensitive Daten in falschen Händen können schwerwiegende Folgen haben, z. B. wenn man an die Entwicklung gefährlicher Waffensysteme denkt. Eine weitere Aufgabe einer sicheren Übertragung sollte sein, dass sich der Absender ohne Zweifel nachweisen läßt.

Daher wird im Rahmen dieser Studienarbeit eine Leitungsverschlüsselung entwickelt und implementiert. Daten, die über eine Leitung versendet werden, sollen während der Übertragung direkt verschlüsselt werden. Die Datenübertragung soll hierbei über eine serielle Schnittstelle des Typs V.24 (RS232-C) ablaufen, da diese Schnittstelle sehr weit verbreitet ist und diese daher an sehr vielen Datenendgeräten zur Verfügung steht.

Neben dem eigentlichen Verschlüsselungsalgorithmus soll dabei insbesondere auch ein Protokoll entworfen werden, welches den Anforderungen entspricht. Anforderungen an das Protokoll sind insbesondere die Flußsteuerung, d. h. die Regelung wie viele Daten dem Leitungsverschlüssler zusammenhängend gesendet werden können, sowie einer Überprüfung der Korrektheit der Daten. Außerdem muss ermittelt werden, inwieweit die Verschlüsselung und das gefundene Protokoll auch auf einem 8-Bit-Mikrocontroller implementierbar sind, so dass der Vorgang in einer akzeptablen Geschwindigkeit abläuft.

Aufgrund des eingesetzten Protokolls ist es zudem erforderlich auch auf dem Datenendgerät eine Software zu implementieren, die mit dem Leitungsverschlüssler kommuniziert. Diese Software soll dem Benutzer, wenn möglich, eine graphische Benutzeroberfläche (GUI) zur Verfügung stellen und unter einem gängigen Betriebssystem (z. B. Windows 9x) ausführbar sein.

1.2. Lösungskonzept

Ein wichtiger Bestandteil der hier vorgestellten Lösung besteht in dem Kommunikationsprotokoll und der Kommunikation zwischen Leitungsverschlüssler und Datenendgerät. Daher gehört es zu den ersten Lösungsschritten zu untersuchen, wie eine solche Kommunikation realisiert werden kann. Hierzu müssen ein Kommunikationsprotokoll entwickelt und dieses in allen an der Kommunikation beteiligten Geräte implementiert werden.

Im nächsten Schritt kann dann der wichtigste Bestandteil einer Leitungsverschlüsselung untersucht werden: Welcher Verschlüsselungsalgorithmus kann genutzt werden und welche Sicherheit bietet dieser? Hier muss die verwendete Hardware-Plattform des Mikrocontrollers sowie die hier zu verwendende Programmiersoftware berücksichtigt werden, welche die Möglichkeiten der realisierbaren Verschlüsselungsalgorithmen erheblich einschränken. Es muss daher ein Kompromiß zwischen Sicherheit und vertretbarem Aufwand gefunden werden.

In den folgenden Schritten können dann Erweiterungen, wie die Schlüsselverwaltung in der auf dem Datenendgerät eingesetzten Kommunikationssoftware, implementiert werden.

2. Der Verschlüsselungsalgorithmus FC1

2.1. Die Funktionsweise des Verschlüsselungsalgorithmus

In dem folgenden Kapitel werden zunächst der für dieses Projekt entwickelte Verschlüsselungsalgorithmus FC1 (Ferres Cipher 1) und dessen mathematisch-theoretischen Grundlagen beschrieben.

Für die Übertragung der Daten wird ein Verschlüsselungsalgorithmus eingesetzt, der eine Weiterentwicklung des sogenannten Vigenère-Chiffre um Operationen, die beispielsweise auch im RC5-Algorithmus verwendet werden, darstellt. Er führt eine symmetrische Verschlüsselung mit einer Schlüssellänge von 48 Bit durch. Dieser Algorithmus ist folgendermaßen aufgebaut:

Zunächst wird mit dem zu verschlüsselnden Byte und dem sechsten Schlüsselbyte eine X-Oder-Operation durchgeführt. Anschließend wird das resultierende Byte um die ebenfalls durch das sechste Schlüsselbyte angegebene Anzahl von Stellen nach links rotiert, wobei allerdings nur die unteren 3 Bits des Schlüsselbytes für diese Rotation relevant sind.

Im nächsten Schritt wird aus den fünf anderen Schlüsselbytes ein aktueller Schlüssel generiert:

Im ersten Schritt wird Schlüsselbyte 1 mit Schlüsselbyte 4 multipliziert.

Im nächsten Schritt werden die Schlüsselbytes 2 und 3 miteinander verglichen und Schlüsselbyte 2 wird um eins erhöht, wenn die beiden identisch sind. Dann wird dieses Byte mit dem Schlüsselbyte 5 multipliziert.

Zur Berechnung des tatsächlichen Schlüssels werden dann die beiden Bytes aus den vorherigen Schritten addiert. Zur eigentlichen Verschlüsselung wird dann dieser resultierende Schlüssel zu dem zu verschlüsselnden Byte addiert.

Anschließend werden das erste und das zweite Schlüsselbyte für die nächste Verschlüsselung jeweils um eins erhöht, wobei sie wieder auf 0 gesetzt werden, wenn sie einen Wert von 256 erreicht haben und somit nicht mehr in einem einzigen Byte gespeichert werden könnten.

Zum Abschluss wird das zu verschlüsselnde Byte mit dem aktuellen ersten Schlüsselbyte x-odert und das zweite Schlüsselbyte wird zu dem Ergebnis addiert.

Das folgende Schaubild soll dieses Verfahren noch einmal verdeutlichen:

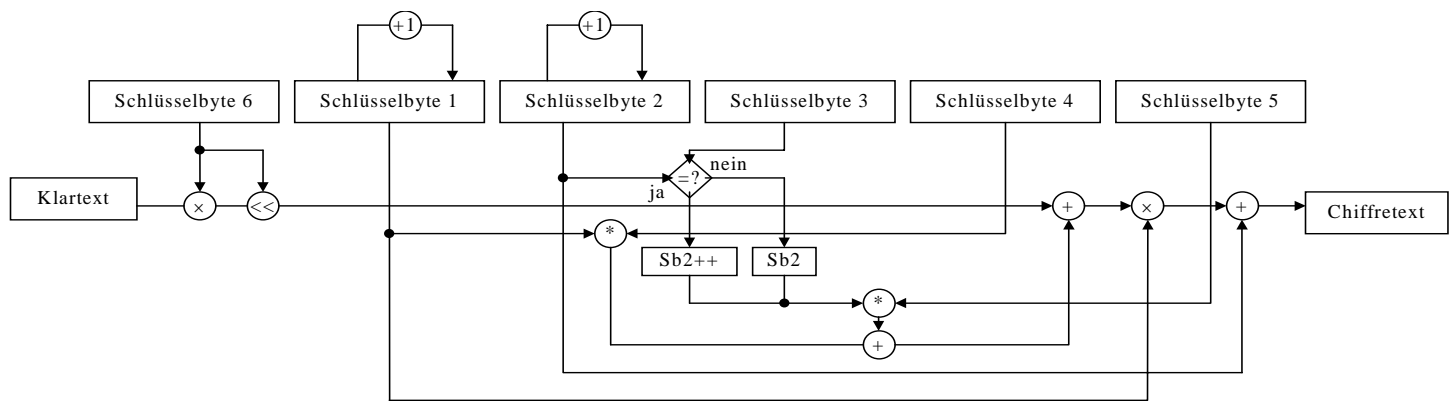


Abb. 2 - Die Verschlüsselung

2.2. Die Entschlüsselung

Das Entschlüsseln durchläuft die Schritte des Verschlüsseln analog in umgekehrter Reihenfolge mit den jeweiligen Umkehroperationen.

Bei den beiden ersten Schritten ist allerdings darauf zu achten, dass die beiden ersten Schlüsselbytes zunächst erhöht werden müssen, bevor das zweite Schlüsselbyte vom verschlüsselten Byte subtrahiert und anschließend zwischen dem Ergebnis und dem ersten Byte eine X-Oder-Operation durchgeführt wird.

Aus diesem Grund sollte der erst im nächsten Schritt benötigte Schlüssel, der exakt auf die gleiche Weise wie beim Verschlüsseln der Daten berechnet wird (Multiplikation des ersten und vierten Schlüsselbytes, Vergleich von zweitem und drittem Schlüsselbyte, Erhöhung des zweiten Schlüsselbytes, wenn diese beiden identisch sind, Multiplikation des sich daraus ergebenden Bytes mit dem fünften Schlüsselbyte und Addition dieses Ergebnisses mit dem Ergebnis der ersten Multiplikation), schon ganz am Anfang eines Entschlüsselungsschrittes berechnet werden, da hier die beiden ersten Schlüsselbytes noch nicht verändert worden sein dürfen. Danach können dann die beiden ersten Schlüsselbytes für die Entschlüsselung des nächsten Bytes und für die ganz am Anfang der Entschlüsselung durchzuführenden Operationen um jeweils eins erhöht werden.

Das berechnete Schlüsselbyte wird nach den schon genannten beiden Schritten von dem Ergebnis dieser Operationen subtrahiert.

Schließlich muss auch noch die Linksrotation um die durch das sechste Schlüsselbyte angegebene Stellenzahl durch eine entsprechende Rotation nach rechts rückgängig gemacht werden.

Als letztes muss das resultierende Byte nochmals mit dem sechsten Schlüsselbyte ver-x-odert werden um auch die erste Operation des Verschlüsselungsalgorithmus wieder rückgängig zu machen.

2.3. Stärken und Schwächen dieses Verschlüsselungsverfahrens

Bei dem hier angewendeten Verfahren handelt es sich nicht um einen Block-, sondern um einen sogenannten Stromchiffre: Jedes Byte wird einzeln ver- oder entschlüsselt.

Um aber dennoch die Häufigkeitsverteilung der verschiedenen Buchstaben zu verschleiern, die charakteristisch für eine bestimmte Sprache ist und die bei einer Verschlüsselung jedes

einzelnen Buchstabens mit dem gleichen Verfahren normalerweise erhalten bleibt, wird eine abgewandelte Form des Vigenère-Chiffre-Verfahrens angewendet:

Dieses Verfahren baut auf dem sogenannten Verschiebe-Chiffre auf, der schon in der Antike von Caesar verwendet wurde. Hierbei wird statt des eigentlichen Buchstabens ein Buchstabe eingesetzt, der immer eine bestimmte Anzahl nach oder vor dem eigentlichen Buchstaben im Alphabet steht. Bei dieser Art der Verschlüsselung bleibt allerdings die Häufigkeitsverteilung der Buchstaben erhalten, da ein bestimmter Buchstabe immer durch den gleichen Geheimbuchstaben ersetzt wird, so dass bei einem deutschen Text sehr schnell erkannt werden könnte, welcher Geheimbuchstabe dem Buchstaben ‚e‘ entspricht, da dieser der in deutschen Texten mit Abstand am häufigsten vorkommende Buchstabe ist. Aus dieser Information lässt sich anhand des Abstandes zwischen dem Buchstaben ‚e‘ und dem zugehörigen Geheimtextbuchstaben dann sehr schnell der Schlüssel berechnen, mit welchem dann auch der Rest des Textes entschlüsselt werden kann.

Diese Häufigkeitsverteilung der Buchstaben wird mit Hilfe des Vigenère-Chiffre verändert. Bei diesem Verfahren wird nicht jeder Buchstabe um die gleiche Stellenzahl verschoben, sondern der jeweils nächste wird um eine andere Anzahl verschoben. Hierzu wird jedem Buchstaben eine Zahl zwischen 0 und 25 zugeordnet. Anschließend wird ein Schlüsselwort gewählt. Die zu den einzelnen Buchstaben dieses Wortes gehörenden Zahlen geben an, um wie viele Stellen der zu verschlüsselnde Buchstabe verschoben werden soll. Der erste Buchstabe des Klartextes wird mit dem ersten Buchstaben des Schlüsselwortes verschlüsselt, der zweite Buchstabe mit dem zweiten des Schlüsselwortes usw. bis das Ende des Schlüsselwortes erreicht ist. Der nächste Buchstabe muss dann wieder mit dem ersten Buchstaben des Schlüsselwortes verschlüsselt werden.

Diese Art der Verschlüsselung bietet allerdings auch noch keine allzu große Sicherheit, da die verwendeten Schlüsselwörter meist relativ kurz sind, oder, sofern es sich um Texte handelt, doch in irgendeiner Sprache geschrieben sind, so dass dort wieder eine Häufigkeitsverteilung der einzelnen Buchstaben zur Analyse verwendet werden könnte.

Das hier angewendete Verfahren erweitert den Vigenère-Chiffre in zweierlei Hinsicht. Zum einen wird an Stelle eines normalen Alphabetes mit nur 26 Buchstaben eine komplette ASCII-Tabelle mit 256-Zeichen verwendet. Daher werden auch Satz- oder Leerzeichen verschlüsselt. Zum anderen werden zur Verschlüsselung nicht Schlüsselwörter oder Texte verwendet, sondern eine aus den einzelnen Schlüsselbytes berechnete Zeichenkette. Diese Zeichenkette entsteht, indem eine komplette ASCII-Tabelle, die an der durch das erste Schlüsselbyte angegebene Stelle beginnt und deren Reihenfolge durch die Multiplikation mit dem vierten Schlüsselbyte verändert wurde, mit einer weiteren ASCII-Tabelle, die an der durch das zweite Schlüsselbyte angegebenen Stelle beginnt, bei welcher ein durch das dritte Schlüsselbyte angegebenes Zeichen fehlt und deren Reihenfolge wiederum durch die Multiplikation mit dem fünften Schlüsselbyte verändert wurde, per Vigenère-Verfahren verschlüsselt wird. Die erste Zeichenkette hat also eine Länge von 256 Zeichen, die zweite eine Länge von 255 Zeichen, so dass das letzte Zeichen der ersten Zeichenkette wieder mit dem ersten Zeichen der zweiten Zeichenkette verschlüsselt wird. Im nächsten Schritt wird dann das erste Zeichen der ersten Zeichenkette mit dem zweiten Zeichen der zweiten Zeichenkette verschlüsselt. So wird also das gleiche Zeichen einer Zeichenkette immer mit einem anderen Zeichen der anderen Zeichenkette verschlüsselt. Erst nach dem Verschlüsseln von

$$255 \cdot 256 = 65280$$

Zeichen wird wieder das erste Zeichen der ersten Zeichenkette mit dem ersten Zeichen der zweiten Zeichenkette verschlüsselt. Die so berechnete Zeichenkette wird nun zum eigentlichen Verschlüsseln des Klartextes per Vigenère-Verfahren verwendet.

Die zusätzlichen Operationen sind erforderlich, um auch eine Verschlüsselung des Klartextes zu gewährleisten, wenn einige Schlüsselbytes relativ ungünstig gewählt werden. So könnten die Schlüsselbytes 4 und 5 den Wert ,0' haben, so dass eine Multiplikation immer den Wert 0 ergibt. Die Addition des Wertes 0 führt jedoch zu keiner Veränderung, so dass der Text unverschlüsselt bliebe. Aus diesem Grund werden die Additions- und die X-Oder-Operationen mit den beiden ersten Schlüsselbytes am Ende der Verschlüsselung durchgeführt. Auch hier werden diese Schlüsselbytes nach jedem Verschlüsselungsvorgang wie bei dem oben beschriebenen Verfahren verändert, um die Häufigkeitsverteilung der Buchstaben zu verschleiern.

Die am Anfang der Verschlüsselung durchgeführte Rotations-Operation mit dem sechsten Schlüsselbyte versucht auch Elemente des bekannten RC5-Verschlüsselungsverfahrens einzubinden. Die zusätzliche X-Oder-Operation mit dem gleichen Schlüsselbyte ist erforderlich, damit bei einer Verschlüsselung, bei der das sechste Schlüsselbyte zum Beispiel einen Wert von 8 hat, ein anderes Resultat bewirkt als bei einer Verschlüsselung, bei der alle anderen Schlüsselbytes gleich sind und lediglich das sechste Schlüsselbyte den Wert 0 oder 16 hat. Würde nur die Rotation durchgeführt, so wäre das Ergebnis in beiden Fällen identisch, da bei der Rotation lediglich Werte zwischen 0 und 7, also die unteren drei Bits des Schlüsselbytes, eine Rolle spielen. Aufgrund der gewählten Reihenfolge der einzelnen Operationen wird erreicht, dass diese nicht ohne weiteres vertauscht werden können.

Die größte Schwäche des hier entwickelten Verfahren ist die geringe Schlüssellänge von nur 48 Bit. Moderne Supercomputer könnten daher den gesamten Schlüsselraum innerhalb von relativ kurzer Zeit durchsuchen, um den Schlüssel mit Hilfe eines Brute-Force-Angriffes zu finden.

Außerdem sind die Schlüssel, deren 4. und 5. Schlüsselbyte den Wert 0 haben, relativ schwach, da hierbei beide Multiplikationen als Ergebnis 0 liefern, so dass auch deren Summe 0 ist und das zu verschlüsselnde Byte durch den Vigenère-Chiffre nicht verändert wird. Daher werden zu Beginn der Verschlüsselung noch die Additions- und X-Oder-Operationen durchgeführt, so dass auch in diesem Fall die Werte der drei ersten Schlüsselbytes nicht unerheblich sind.

2.4. Ein Brute-Force-Angriff auf den FC1-Algorithmus

Ein wichtiges Kriterium zur Beurteilung der Sicherheit eines Verschlüsselungsalgorithmus ist die Zeitdauer, die mit Hilfe eines Brute-Force-Angriffes benötigt wird um einen Schlüssel zu finden. Bei dieser Art des Angriffs wird weder versucht aus den verschlüsselten Daten irgendwelche Informationen über den verwendeten Schlüssel zu gewinnen, noch muss ein bekanntes Paar aus verschlüsseltem und unverschlüsseltem Text bekannt sein, was bei einer sogenannten Know-Plaintext-Attacke notwendig ist um den korrekten Schlüssel zu finden. Bei dieser Attacke werden der Reihe nach alle möglichen Schlüssel ausprobiert, bis der passende gefunden wurde. Um zu erkennen wann der passende Schlüssel gefunden wurde, ist meist jedoch trotzdem die Kenntnis wenigstens eines geringen Teils des unverschlüsselten Textes nötig, damit der Entschlüsselungsprozess automatisch ablaufen kann und nicht nach jeder erfolgten Entschlüsselung ein Mensch überprüfen müsste, ob ein korrekter Klartext gewonnen wurde.

Der Verschlüsselungsalgorithmus FC1 verwendet eine Schlüssellänge von 48 Bit, was einem Schlüsselraum von

$$2^{48} = 281.474.976.710.656 \text{ (ca. 281,5 Billionen)}$$

Schlüsseln entspricht. So gewaltig diese Menge erscheinen mag, ist es durch den Einsatz von massiv paralleler Berechnung, z. B. durch Verteilung von unterschiedlichen Schlüsseln über das Internet und Durchführung der Entschlüsselung auf tausenden von ans Netz angeschlossenen Rechnern, wie dies beispielsweise in den Projekten von distributed.net gemacht wird, möglich, den gesamten Schlüsselraum in relativ kurzer Zeit zu durchsuchen. So gelang es distributed.net eine mit dem Algorithmus DES (56 Bit Schlüssellänge) verschlüsselte Nachricht innerhalb von 24 Stunden zu knacken.

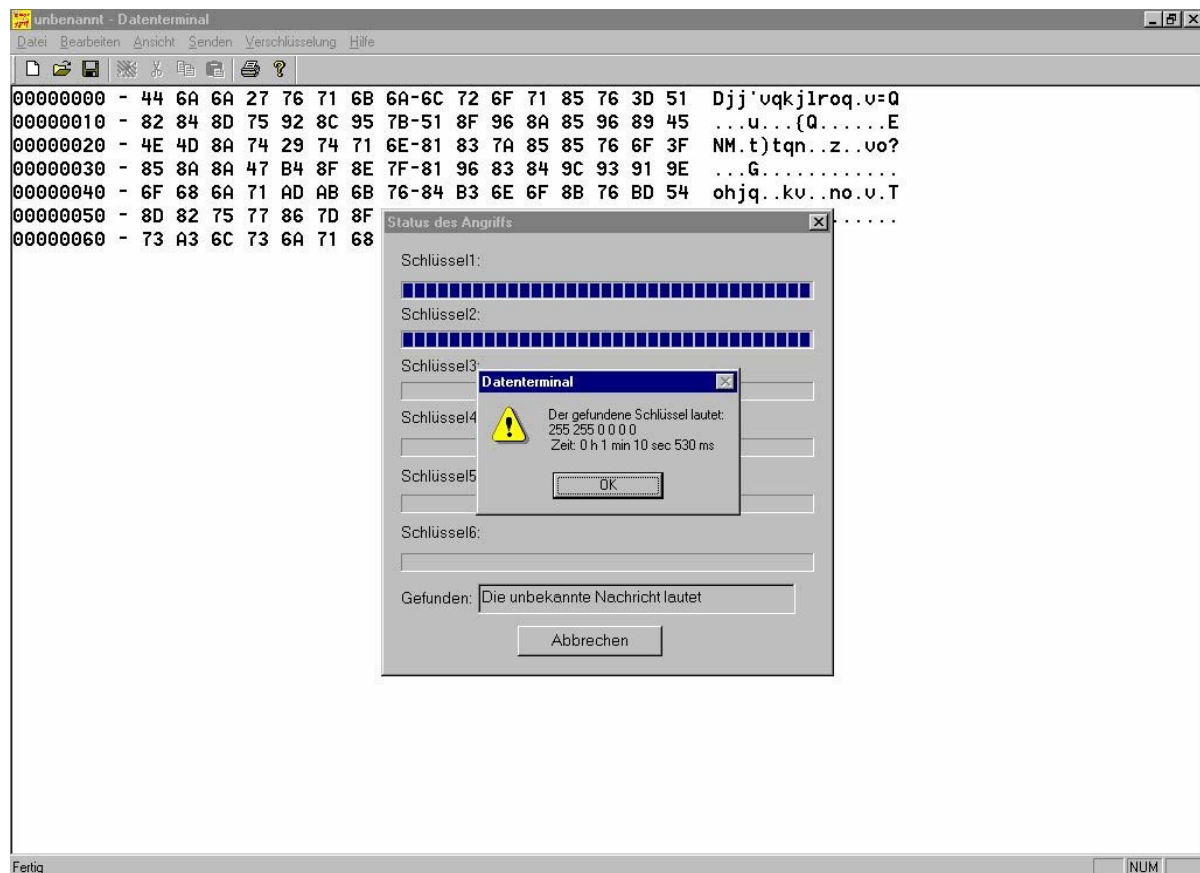


Abb. 3 - Ein Brute-Force-Angriff mit dem Programm 'Datenterminal'

Steht jedoch nur ein einzelner Rechner zu Verfügung um den gesamten Schlüsselraum zu durchsuchen, beispielsweise ein normaler PC, so benötigt dieser doch sehr viel Zeit zum Durchsuchen des gesamten Schlüsselraumes. Das innerhalb dieses Projektes entwickelte Programm ‚Datenterminal‘ (siehe Kapitel 6. – „Das Windows-Programm ‚Datenterminal‘“) bietet die Möglichkeit, einen solchen Brute-Force-Angriff durchzuführen. Es fordert den Benutzer auf einen Teil der Nachricht in unverschlüsselter Form anzugeben und sucht anschließend selbständig nach einem passenden Schlüssel, wobei für den Benutzer der aktuelle Fortschritt auf dem Bildschirm angezeigt wird. Nachdem ein möglicher Schlüssel gefunden wurde, wird dieser dem Benutzer angezeigt und auch die Zeit, die zum Finden dieses Schlüssels benötigt wurde, wird ausgegeben. Ein Test mit der Nachricht:

Die unbekannte Nachricht lautet:

Es klapperten die Klapperschlangen, bis ihre Klappern schlapper klangen.

zeigt, dass ein Pentium-II-300-Rechner für 65.536 Entschlüsselungen eine Zeit von 1 Minute und 10 Sekunden benötigt. Rechnet man diese Zeit hoch, so würde ein solcher Rechner für die Durchsuchung des gesamten Schlüsselraumes eine Zeit von mehr als 9.526 Jahren benötigen. Allerdings kann davon ausgegangen werden, dass ein großer Teil der Zeit hierbei für die Statusanzeige auf dem Bildschirm benötigt wird und die reine Entschlüsselung weniger Zeit in Anspruch nehmen würde.

3. Das Kommunikationsprotokoll FCP

In diesem Kapitel wird die Theorie und der Ablauf des eigens für dieses Projekt entwickelte Kommunikationsprotokolls FCP (Ferres Communication Protocol) beschrieben.

In der Datenübertragung können Daten nicht ohne bestimmte Regeln über eine Leitung vom Sender zum Empfänger geschickt werden. Die beteiligten Parteien müssen sich zuvor darauf geeinigt haben, wie diese Kommunikation ablaufen soll. Sie legen also einen Rahmen fest, der in der Datenkommunikation als Protokoll bezeichnet wird. Verschiedene bekannte Protokolle sind beispielsweise TCP/IP, IPX u. a.

Der Hauptgrund für den Einsatz eines Protokolls in dem vorliegenden Projekt ist die Flusststeuerung: Die Verschlüsselungsschaltung soll in bestimmte Zustände versetzt werden können um die Daten jeweils korrekt zu verarbeiten. Außerdem soll eine Prüfsumme gebildet werden, die feststellt, inwiefern sich Bits während der Übertragung geändert haben.

In dem FCP-Protokoll werden folgende Schritte unterschieden:

1. Schlüsselübertragung
2. Übertragung der Länge der zu sendenden Daten
3. Übertragung der Daten
4. Übertragung einer Endekennung

Im folgenden sollen die einzelnen Schritte der Übertragung näher erläutert werden.

3.1. Die einzelnen Schritte der Übertragung

a) Übertragung des Schlüssels

Für die momentan implementierte Verschlüsselung wird ein 48-Bit-Schlüssel (siehe Kapitel 2. – „*Der Verschlüsselungsalgorithmus FCI*“) verwendet. Dieser muss als erstes in den Leitungsverchlüsseler übertragen und dort gespeichert werden. Insgesamt werden jedoch 7 Bytes (56 Bit) übertragen, da als letztes Byte eine Prüfsumme übertragen wird. Das Prüfbyte wird aus der Summe der übertragenen Schlüsselbytes gewonnen (siehe Kapitel 3.2. – „*Die Prüfsumme und das Acknowledge*“).

b) Übertragung der Datenlänge

Vor Beginn jeder Übertragung wird die Länge der insgesamt zu übertragenden Daten vom Sender zum Empfänger geschickt. Dies ist notwendig, damit auch der Empfänger den Benutzer über den Fortschritt der Übertragung informieren kann. Außerdem muss auf der Empfängerseite ein entsprechend großer Puffer angelegt werden, der die Daten aufnimmt.

Obwohl für die Länge lediglich 4 Bytes benötigt werden, werden insgesamt 8 Bytes übertragen, die übrigen Bytes werden mit dem Wert 0 gefüllt. Als neuntes Byte wird schließlich ebenfalls ein Prüfbyte übertragen, welches wie beim Schlüssel aus der Summe der Datenbytes gebildet wird (siehe Kapitel 3.2. – „*Die Prüfsumme und das Acknowledge*“).

c) Übertragung eines Datenblocks

Die Daten werden in 8-Byte-Blöcken übertragen. Ist die gesamte Datenlänge nicht ein Vielfaches von 8 Byte, so muss der letzte Block mit der entsprechenden Anzahl von Bytes, die den Wert 0 enthalten, aufgefüllt werden. Auch bei der Übertragung der Daten wird als

neuntes Byte ein Prüfbyte gesendet. Es wird analog zu dem Prüfbyte der Längenübertragung gebildet.

Ein Datenframe hat also folgenden Aufbau:

Daten- byte1	Daten- byte2	Daten- byte3	Daten- byte4	Daten- byte5	Daten- byte6	Daten- byte7	Daten- byte8	Prüf- byte
-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------------

Abb. 4 – Aufbau eines Datenframes

d) Übertragung der Endekennung

Die Endekennung, die zur Kennzeichnung des Endes einer Übertragung gesendet wird, besteht ebenfalls aus einer Sequenz von 9 Bytes, wobei 8 Bytes den Wert 0 haben, das neunte Byte den Wert 04h (=‘EOT‘), so dass dieser Block für einen Datenblock gehalten wird, bei welchem die Prüfsumme, die während des Empfangs der Datenbytes berechnet wird, nicht der Prüfsumme entspricht, die tatsächlich am Ende der Daten gesendet wird. Die Summe der acht Bytes mit dem Wert 0 ergibt ebenfalls 0, als Prüfsumme wird jedoch ein ‚EOT‘ (Wert ‚04h‘) gesendet. Durch diese bewusste Protokollverletzung kann das Ende der Übertragung erkannt werden. Ein einfaches Senden einer bestimmten Sequenz mit korrekter Prüfsumme kann hierfür nicht verwendet werden, da diese Sequenz ja auch inmitten der Daten vorkommen könnte, so dass die Übertragung schon zu früh abgebrochen werden würde, wenn eine solche Sequenz das Ende der Übertragung markieren würde.

Ein Problem könnte allerdings ein Datenframe darstellen, bei dem ebenfalls alle Datenbytes den Wert ‚0‘ haben, im Prüfsummenbyte jedoch das dritte Bit gekippt ist und es somit den Wert ‚04h‘ hat. In einem solchen Fall würde dieser Datenframe für eine Endekennung und nicht für einen fehlerhaften Datenframe gehalten werden.

Die Endekennung hat somit folgenden Aufbau:

0	0	0	0	0	0	0	0	EOT
---	---	---	---	---	---	---	---	------------

Abb. 5 – Die Endekennung

3.2. Die Prüfsumme und das Acknowledge

Die Prüfsumme besteht jeweils aus der Summe der zuvor gesendeten 8 Datenbytes, allerdings kann sie den Wert 255 nicht überschreiten, da sonst das Ergebnis nicht mehr in einem einzigen Byte gespeichert werden könnte. Wird dieser Wert bei der Addition überschritten, so ergibt sich ein Überlauf. Werden z. B. die Werte ‚84‘, ‚104‘, ‚111‘, ‚114‘, ‚115‘, ‚116‘, ‚101‘ und ‚110‘ gesendet, so ergibt sich eigentlich ein Wert von ‚955‘, in einem Byte können allerdings nur die unteren 8 Bit dieses Wertes, also der Wert ‚187‘ gespeichert werden. Daher wird dieser Wert in das angehängte Prüfbyte geschrieben. Bei dieser Addition wird zwar das Carry-Flag gesetzt, dieses wird jedoch nicht beachtet.

Ein solcher Überlauf muss jedoch nicht weiter berücksichtigt werden, da er auch bei einer erneuten Berechnung durch den Empfänger auftritt, so dass diese Rechnung dennoch zum gleichen Ergebnis führt.

Das Protokoll ist so aufgebaut, dass der Sender solange mit dem Senden neuer Daten wartet, bis er vom Empfänger ein Acknowledge (Wert ‚06h‘) für die zuletzt gesendeten Daten erhalten hat.

Handelt es sich bei dem Empfänger um die Verschlüsselungsschaltung, so berechnet diese die Prüfsumme und sendet sofort ein negatives Acknowledge (Wert ,15h'), wenn diese nicht mit der empfangenen Prüfsumme übereinstimmt. Stimmen die beiden Werte überein, so werden die Daten ver- bzw. entschlüsselt, dazu die entsprechende Prüfsumme gebildet und diese Daten werden weiterversendet. Auch hier wird auf eine positive Rückmeldung gewartet. Wird diese empfangen, so wird auch ein Acknowledge-Byte zum ursprünglichen Sender geschickt.

Erhält der Sender statt dessen ein negatives Acknowledge, so sendet er die zuvor gesendeten Daten noch einmal und der gesamte Ablauf beginnt erneut. Dies soll in folgendem Schaubild verdeutlicht werden:

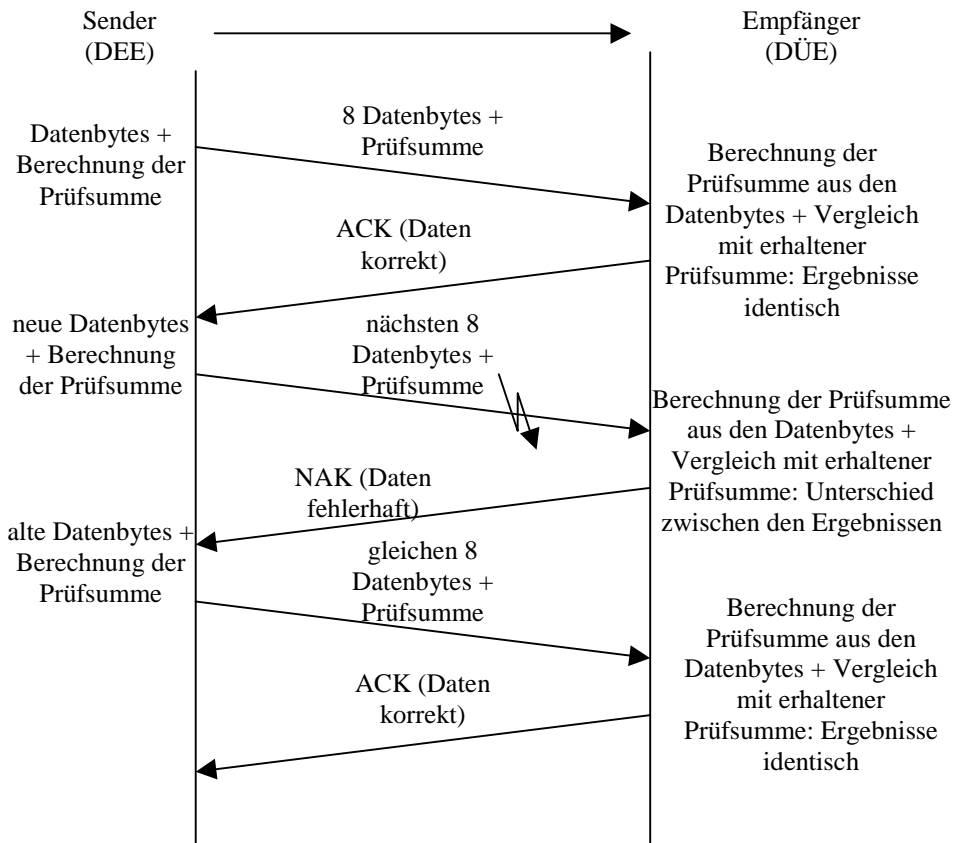


Abb. 6 - Versenden von Datenpaketen

3.3. Bewertung des Protokolles

Bei dem FCP handelt es sich um ein Halbduplex-Protokoll, bei welchem die Übertragung der Daten immer nur in eine Richtung erfolgen kann. Der Leitungsverchlüsseler entscheidet sogar anhand der Übertragungsrichtung, ob die Daten ver- oder entschlüsselt werden sollen.

Aus diesem Grund und da auch für die Berechnung der Prüfsumme kein sehr komplexes Verfahren wie CRC eingesetzt wird, kann der Aufbau des Protokolles sehr einfach gehalten werden. Nachteil des Verfahren sind allerdings die fehlende Sicherheit, da mit Hilfe der Prüfsumme Übertragungsfehler zwar erkannt, nicht jedoch korrigiert werden können. Es gibt sogar Übertragungsfehler, die überhaupt nicht erkannt werden. Beispielsweise bleibt die Prüfsumme gleich, wenn innerhalb eines Bytes ein Bit von 1 nach 0 kippt, das gleiche Bit aber innerhalb eines anderen Bytes des gleichen Datenframes von 0 nach 1 kippt. Auch die Enderkennung ist, wie bereits erwähnt, nicht gegen Übertragungsfehler abgesichert.

3.4. Der Gesamtablauf des Protokolls

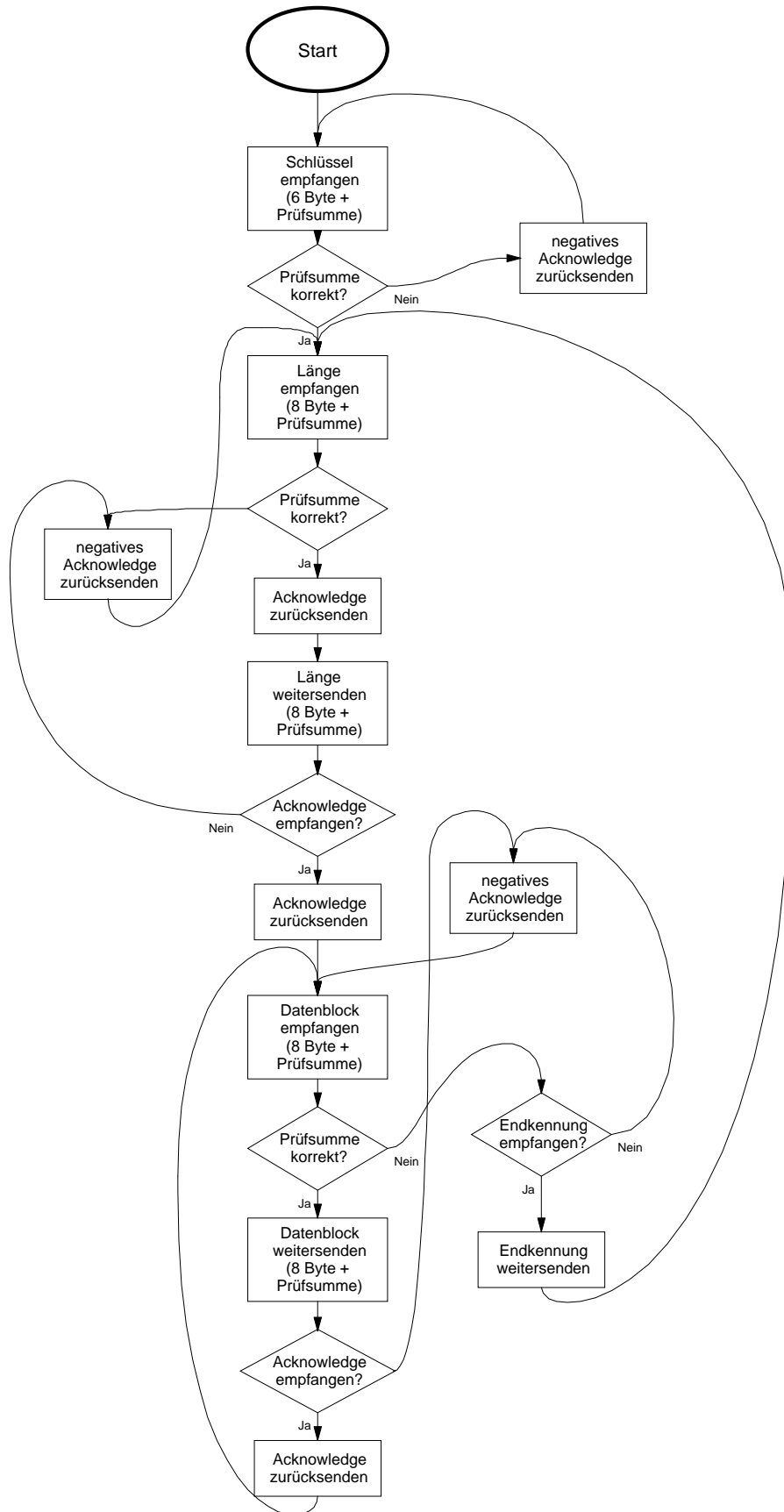


Abb. 7 - Der Kommunikationsablauf

4. Verwendete Werkzeuge

Nachdem in den vorangegangenen Kapiteln die theoretischen Grundlagen für die Leitungsverchlüsselung erläutert wurden, wird nun deren Umsetzung beschrieben. In diesem Kapitel werden zunächst alle zur Realisierung des Projektes verwendeten Werkzeuge beschrieben.

4.1. Die eingesetzte Hardware

Zur Realisierung der eigentlichen Verschlüsselung wird eine Schaltung eingesetzt, die einen 8-Bit-Mikrocontroller des Typs Siemens 80C537 enthält. Es handelt sich dabei um einen Controller, der zur i8051-Familie gehört, allerdings weitaus leistungsfähiger ist als das Original. Bei der Realisierung des hier vorgestellten Projektes spielt insbesondere dessen Eigenschaft, über zwei serielle Schnittstellen zu verfügen, eine wichtige Rolle. Außerdem sind Mikrocontroller der i8051 sehr weit verbreitet, für viele Zwecke einsetzbar und lassen sich relativ einfach programmieren. Eingesetzt wird daher die Schaltung LAB537 von Sven Rakers [Ra1999], welche mit einem mit 16 MHz getakteten Mikrocontroller des genannten Typs ausgestattet ist.

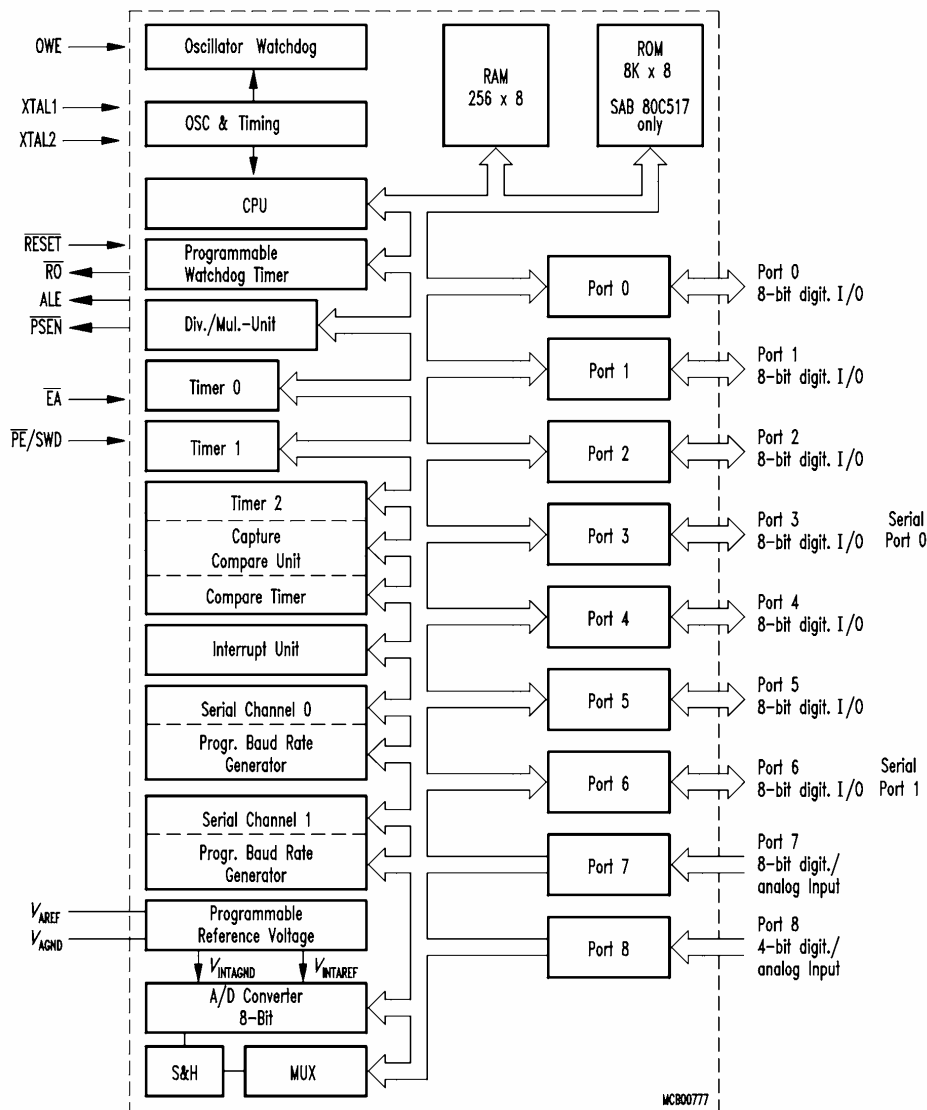


Abb. 8 - Blockschaltbild des MC80C537 (aus [MüWa1999])

Soll die Schaltung mit einem Eprom betrieben werden, so wird zusätzlich noch ein Eprom des Typs 27C512 benötigt, auf welches das im Kapitel 5 beschriebene Mikrocontroller-Programm gebrannt wird. Dieser Baustein ersetzt dann das Eprom, auf welchem sich bei der Original-Schaltung das CESY-Betriebssystem befindet.

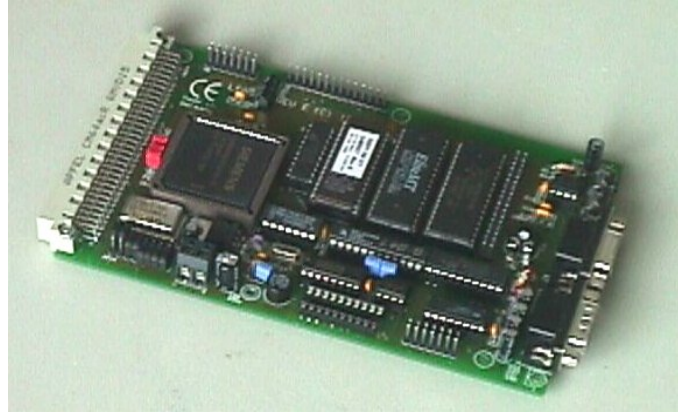


Abb. 9 - Die Mikrocontrollerschaltung

Als Datenendgerät werden Personal Computer eingesetzt, die über deren serielle Schnittstelle mit der Schaltung kommunizieren.

4.2. Zur Entwicklung eingesetzte Software

Zur Programmierung der Mikrocontrollerschaltung wird die Entwicklungsumgebung „CESY“ professional für Windows in der Version 7.3 (Rev A 14-09-99) eingesetzt. Außerdem befindet sich auf der Schaltung ein Eprom, welches das Betriebssystem „CESY“ enthält, das u. a. auch dafür eingesetzt werden kann um die Schaltung als Eprom-Simulator zu betreiben. Diese Eigenschaft ist insbesondere bei der Entwicklung des Mikrocontroller-Programmes hilfreich.

Das Empfangsprogramm „Datenterminal“ (siehe Kapitel 6), welches auf einem PC unter den Betriebssystemen Windows 95 und Windows 98 lauffähig ist, wird mit dem MS Visual C++ Compiler (Version 6.0) erstellt.

Zum Erstellen einer Installationsroutine für dieses Programm wird die Freeware „Inno Setup“ von Jordan Russell in der Version 1.2.14 eingesetzt.

5. Das Mikrocontroller-Programm

Das hier beschriebene Projekt besteht aus zwei Bestandteilen: Auf der einen Seite ist dort die auf dem Datenendgerät eingesetzte Kommunikationsanwendung, mit welcher der Benutzer meist direkt arbeitet, auf der anderen ist dort der Leitungsverchlüsseler, der als Hardware zwischen das Datenendgerät und die Übertragungsstrecke geschaltet wird. Auch dieser benötigt eine spezielle Software, die zu diesem Zweck entwickelt werden mußte und im folgenden Kapitel beschrieben wird.

Das Mikrocontroller-Programm ist in 80C537-Assembler geschrieben. Es besteht aus einem Hauptprogramm, welches hauptsächlich für die Einhaltung des Übertragungsprotokolls zuständig ist, und verschiedenen Unterprogrammen, die die Initialisierung der Schnittstellen, die Übertragung und den Empfang einzelner Bytes sowie die Verschlüsselung der Daten durchführen.

5.1. Das Hauptprogramm

Das Hauptprogramm ist so aufgebaut, dass nach einem HW-Reset des Mikrocontrollers zunächst einmal die beiden seriellen Schnittstellen initialisiert werden. Hierzu werden die beiden Unterprogramme ‚s1init‘ und ‚s2init‘ aufgerufen.

Anschließend wird ein Programmteil ausgeführt, der auf den Empfang von sechs Schlüsselbytes sowie des zugehörigen Acknowledge-Bytes wartet. Es muss also nach einem Reset der Mikrocontrollerschaltung bzw. nach Aktivierung der Schaltung immer zuerst ein Schlüssel in den Mikrocontroller übertragen werden, bevor weitere Daten übertragen werden können. Dieser Schlüssel wird im internen RAM des Mikrocontrollers gespeichert.

Nachdem ein Schlüssel empfangen wurde, wird ein Flag-Bit gesetzt, welches den Beginn einer Übertragung kennzeichnet. Dies ist erforderlich, da die ersten acht Bytes die Länge der zu übertragenden Daten enthalten und daher nicht verschlüsselt werden dürfen. Das Mikrocontroller-Programm selbst wertet diese Information allerdings nicht aus, lediglich das Programm am Datenendgerät (PC) generiert hieraus eine entsprechende Information für den Benutzer und legt einen der Datenlänge entsprechenden Puffer für die Daten an. Des Weiteren werden hier die beiden Flags, die einen Empfang der Daten an einem der beiden seriellen Ports anzeigen, ebenfalls gelöscht.

Es folgt nun die Hauptschleife des Mikrocontroller-Programmes. An deren Anfang steht eine weitere Schleife, die acht Bytes, die von einem der beiden seriellen Ports empfangen werden, in einen im internen RAM des Mikrocontrollers liegenden Pufferspeicher schreibt. Zudem wird gleichzeitig die zugehörige Prüfsumme berechnet. Außerdem wird ein Flag-Bit gesetzt, welches kennzeichnet, an welchem seriellen Port Daten empfangen wurden, damit das Programm im weiteren Verlauf erkennen kann, ob diese Daten ver- oder entschlüsselt werden sollen und an welcher seriellen Schnittstelle die bearbeiteten Daten ausgegeben werden sollen. Daten, die an der ersten seriellen Schnittstelle empfangen werden, müssen verschlüsselt und an der zweiten seriellen Schnittstelle ausgegeben werden, während Daten, die am zweiten seriellen Port empfangen werden, entschlüsselt und am ersten seriellen Port ausgegeben werden müssen.

Nach dem Empfang des neunten Bytes wird dieses Byte mit der während der Übertragung berechneten Prüfsumme verglichen. Stimmen diese beiden Werte überein, so wird das Programm an der Stelle fortgesetzt, an welcher die Daten verschlüsselt werden. Sind die

Werte nicht identisch, so wird zunächst geprüft, ob es sich um eine Endkennung handelt. Ist dies der Fall, so wird diese sofort weitergesendet. In allen anderen Fällen wird sofort ein negatives Acknowledge (,NAK' = ,15h') an den Sender geschickt.

Bevor die Daten weiterversendet werden, müssen sie ver- bzw. entschlüsselt, und es muss die hierzu passende Prüfsumme berechnet werden. Hierzu wird ein Unterprogramm aufgerufen, welches die Daten aus dem Empfangspuffer liest, verschlüsselt bzw. entschlüsselt und einschließlich der neuen Prüfsummen in einen weiteren Puffer, den Sendepuffer, kopiert.

Anschließend werden diese Daten aus dem Sendepuffer gelesen und an der anderen seriellen Schnittstelle ausgegeben. Nach dem Versenden der Daten wird auf das Acknowledge-Byte des Empfängers gewartet. Wenn ein solches Byte empfangen wurde, so kann auch ein Acknowledge-Byte an den ursprünglichen Sender der Daten geschickt werden und er kann weitere Daten senden. Empfängt die Mikrocontrollerschaltung an dieser Stelle ein negatives Acknowledge, so wird auch dieses an den ursprünglichen Sender der Nachricht weitergereicht. In beiden Fällen kehrt das Programm jedoch zum Anfang der Hauptschleife zurück. Diese ist eine Endlosschleife, so dass nur nach einem Reset der Schaltung ein neuer Schlüssel geladen werden kann.

5.2. Die Unterprogramme zur seriellen Übertragung

Für die serielle Übertragung gibt es insgesamt acht verschiedene Unterprogramme, wobei jeweils zwei die gleiche Funktionalität, jeweils für einen der beiden seriellen Ports des Mikrocontrollers, erfüllen. Es sind jeweils eigene Unterprogramme für die beiden seriellen Ports erforderlich, weil das Programm in Assembler geschrieben ist und somit die Register direkt angesprochen werden. Jeder Port wird jedoch über eigene Register angesprochen. Außerdem ist die Generierung der Baudrate für die beiden Ports unterschiedlich.

Zunächst müssen die seriellen Schnittstellen initialisiert werden. Das bedeutet, dass der Empfangsmode und die Baudrate eingestellt werden müssen.

Für die erste serielle Schnittstelle geschieht dies durch die Routine ,s1init'. Es wird der Modus 1 (siehe [MüWa1999]) eingestellt, bei dem acht Datenbits und ein Stoppbit asynchron übertragen werden (UART 8-Bit). Außerdem wird ein Empfang von Daten ermöglicht. Diese Einstellungen werden durch das Setzen von Bits im Register S0CON durchgeführt.

Des Weiteren muss die Baudrate eingestellt werden. Hierzu wird bei der ersten Schnittstelle der Timer 1 genutzt, in dem der nach der Formel

$$\text{Baudrate} = \frac{f_{osz}}{192 \cdot (256 - Th1)} \Rightarrow Th1 = 256 - \frac{f_{osz}}{192 \cdot \text{Baudrate}} \quad [\text{MüWa1999}]$$

berechnete Wert in das Timer-Register Th1 geschrieben wird und der Timer im Autoreload-Modus betrieben wird. Bei diesem Mode wird der Timer als 8-Bit Zähler betrieben und nach jedem Überlauf des T11-Registers der Wert des Th1-Registers in T11 geladen. Zum Einstellen dieses Modus müssen in weiteren Flag-Registern Bits ein- bzw. ausgeblendet werden.

Auch bei der zweiten seriellen Schnittstelle, welche durch die Routine ,s2init' initialisiert wird, werden die Bits im S1CON-Register so gesetzt, dass acht Bits und ein Stoppbit

asynchron übertragen werden können. Das Bit, welches den Empfang von Daten ermöglicht, wird ebenfalls gesetzt.

Zum Festlegen der Baudrate wird hier allerdings der interne Baudratengenerator verwendet, bei dem der nach der Formel

$$Baudrate = \frac{f_{osz}}{32 \cdot (256 - relwert)} \Rightarrow relwert = 256 - \frac{f_{osz}}{32 \cdot Baudrate} \quad [\text{MüWa1999}]$$

berechnete Wert in das S1REL-Register geschrieben werden muss.

Für beide Schnittstellen wird in diesem Projekt eine Baudrate von 9600 Baud gewählt, da dieser Wert aufgrund des für die Erzeugung einer Baudrate ungünstigen Quarz-Taktes von 16 MHz am besten eingestellt werden kann. In den beiden obigen Formeln ergibt sich für den ‚relwert‘ jeweils eine Kommazahl, es kann jedoch nur ein Integerwert in die jeweiligen Register geschrieben werden, so dass die eigentliche Baudrate leicht von dem gewünschten Wert abweicht. So gilt für die mit Hilfe des Baudratengenerators erzeugte Baudrate des zweiten seriellen Ports:

$$relwert = 256 - \frac{f_{osz}}{32 \cdot Baudrate} = 256 - \frac{16MHz}{32 \cdot 9600Baud} = 203,916 \approx 204$$

aber:

$$Baudrate = \frac{f_{osz}}{32 \cdot (256 - relwert)} = \frac{16MHz}{32 \cdot (256 - 204)} = 9615,38Baud$$

Zum Empfangen eines Datenbytes werden die beiden Routinen ‚wait_receivesX‘ und ‚receivesX‘ benötigt (‚X‘ steht jeweils für die Nummer der seriellen Schnittstelle). Die erste Routine prüft, ob ein Flag-Bit im SCON-Register der jeweiligen seriellen Schnittstelle gesetzt wurde, welches den vollständigen Empfang eines Datenbytes anzeigt. Ist dieses Flag gesetzt, so wird auch das zugehörige programminterne Flag zum Anzeigen des Empfangs eines Bytes an einem seriellen Port gesetzt. Der aufrufende Programmteil muss also dieses Flag prüfen, um zu erkennen, ob ein Byte empfangen wurde oder nicht.

Die zweite Routine kopiert nun das empfangene Byte aus dem entsprechenden SBUF-Register in den Akku und setzt das SCON-Register wieder zurück. Dieses Register muss nochmals komplett mit dem gleichen Wert gesetzt werden, der auch beim Initialisieren der Schnittstelle in dieses Register geschrieben wurde. Ein Ausblenden des Flag-Bits, welches den Empfang der Daten angezeigt hat, reicht nicht aus (z. B. mittels einer Bit-Maske und einer AND-Verknüpfung). Auch das programminterne Flag-Bit, welches den Empfang des Datenbytes angezeigt hat, wird an dieser Stelle wieder gelöscht. Der aufrufende Programmteil kann das empfangene Byte aus dem Akku lesen und weiterverarbeiten.

Zum Versenden eines Bytes wird für jede Schnittstelle jeweils nur eine Routine (‚sends1‘ und ‚sends2‘) benötigt. Vor dem Aufruf dieses Unterprogrammes muss das zu versendende Byte in den Akku geschrieben werden. Die Routine kopiert dieses Byte aus dem Akku in das entsprechende SBUF-Register und wartet so lange, bis ein Flag-Bit in dem zugehörigen SCON-Register gesetzt ist. Hierdurch wird angezeigt, dass das komplette Byte versendet wurde. Auch hier wird das SCON-Register vor Verlassen der Routine wieder auf seinen ursprünglichen Wert zurückgesetzt.

5.3. Die Routinen zur Ver- bzw. Entschlüsselung der Daten

Zum Ver- bzw. Entschlüsseln der Daten werden die vier Unterprogramme ‚`cryption`‘, ‚`encrypt`‘, ‚`decrypt`‘ und ‚`makekey`‘ eingesetzt.

Das Unterprogramm ‚`cryption`‘ ist dafür zuständig, alle Daten, die empfangen wurden, aus dem Empfangspuffer zu lesen, die Routinen zur Ver- bzw. Entschlüsselung aufzurufen und die bearbeiteten Daten in den Sendepuffer zu kopieren. Außerdem wird hier auch die Prüfsumme neu berechnet und als neuntes Byte ebenfalls in den Sendepuffer geschrieben, da sich dieses Byte durch das Verschlüsseln der Daten ebenfalls ändert.

Die beiden schon erwähnten Routinen ‚`encrypt`‘ und ‚`decrypt`‘ werden zum eigentlichen Ver- bzw. Entschlüsseln jeweils eines einzelnen Datenbytes verwendet. Die Routine ‚`encrypt`‘ implementiert hierbei den Verschlüsselungsalgorithmus, ‚`decrypt`‘ den Entschlüsselungsalgorithmus. Diese ist genau symmetrisch aufgebaut, da es sich um eine symmetrische Verschlüsselung handelt. Wie dieser Algorithmus aufgebaut ist, wird im Kapitel 2 ‚Der Verschlüsselungsalgorithmus‘ beschrieben. Außerdem wird in diesen Routinen jeweils das Flag, welches die erste Übertragung kennzeichnet, geprüft, um diese von der Ver- bzw. Entschlüsselung auszuschließen.

Die Routine ‚`makekey`‘ des Mikrocontroller-Programmes, welche ebenfalls Aufgaben der Ver- bzw. Entschlüsselung übernimmt, ist für die Berechnung des für diese Operation einzusetzenden Schlüssels zuständig. Diese Routine wird auch dann aufgerufen, wenn es sich um den ersten 8-Byte-Block der Übertragung handelt, der nicht verschlüsselt werden soll. In diesem Fall werden die beiden ersten Schlüsselbytes (siehe Abb.2 – Die Verschlüsselung) in Register des Mikrocontrollers geladen, so daß diese für die spätere Bearbeitung manipuliert werden können ohne den ursprünglichen Schlüssel zu vernichten, welcher im internen RAM des Mikrocontrollers gespeichert ist. Als aktueller Schlüssel wird bei der Übertragung der Länge, also des ersten Datenframes (siehe Kapitel 3.1. – *„Die einzelnen Schritte der Übertragung“*) der Wert ‚0‘ verwendet, der keine Veränderung der Daten bewirkt, da dieser Wert zu den Daten addiert wird.

5.4. Der logische Ablauf des Mikrocontroller-Programmes

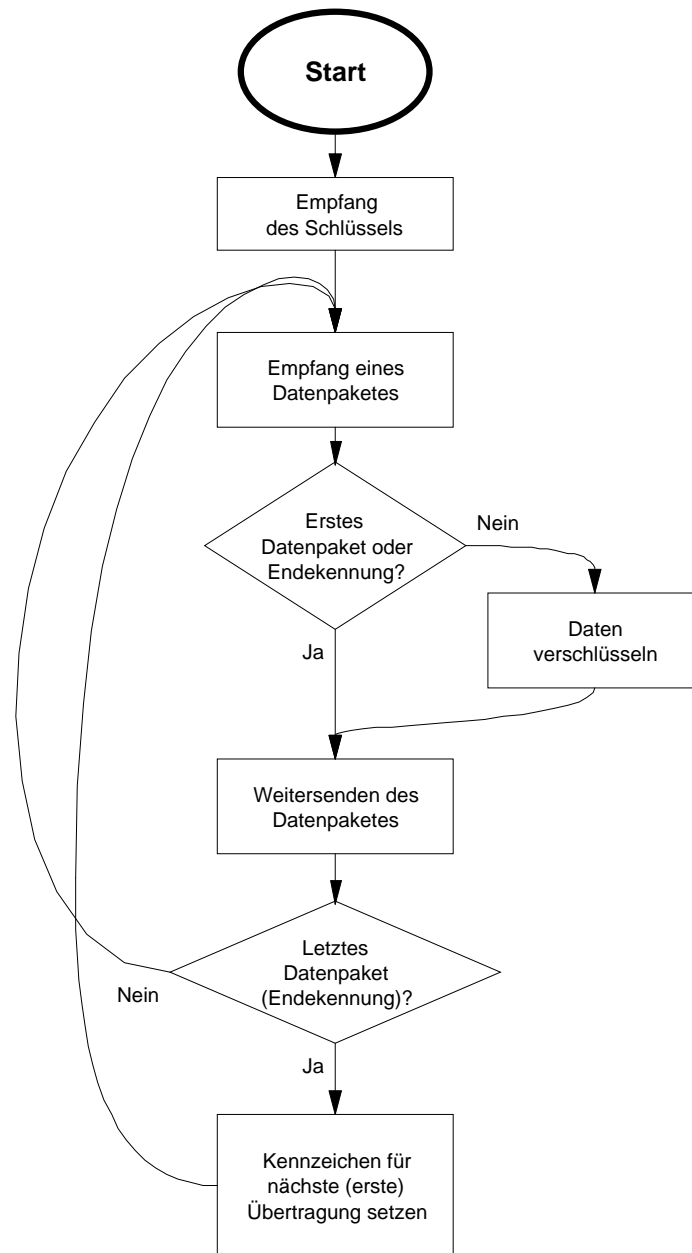


Abb. 10 - Der Ablauf des Mikrocontroller-Programmes

6. Das Windows-Programm ‚Datenterminal‘

Neben der Mikrocontrollerschaltung findet die Software ‚Datenterminal‘ Verwendung, die mit Hilfe des MS Visual C++ Compilers in der Version 6.0 erstellt wurde und ebenfalls das zur Kommunikation verwendete Protokoll beinhaltet. Außerdem wird diese Software verwendet, um auch verschlüsselte Daten empfangen oder senden zu können, um eine zweite Schaltung bei Durchführung des Projektes überflüssig zu machen. Des Weiteren ist dieses Programm in der Lage, Daten sofort bei Eingabe eines Schlüssels nach dem im zweiten Kapitel beschriebenen Verschlüsselungsverfahren zu ver- oder entschlüsseln.

6.1. Aufbau des Programmes ‚Datenterminal‘

Das Programm ‚Datenterminal‘ ist in der objektorientierten Programmiersprache C++ implementiert. Es werden also verschiedene Klassen verwendet, um die Funktionalität zur Verfügung zu stellen. Des Weiteren finden auch die Klassen der MFC Verwendung, die für eine vereinfachte Programmierung unter Windows von Microsoft zur Verfügung gestellt werden.

Das Programm basiert auf dem Programm ‚HexView‘, welches in [LeAr1998] vorgestellt wird. Allerdings beinhaltet die hier verwendete Version einen erheblich erweiterten Funktionsumfang. Des Weiteren sind Fehler, die bei der Ausgabe von Daten auf dem Bildschirm mit der Originalversion dieses Programmes entstehen, soweit wie möglich behoben. Bei dieser Version gibt es das Problem, dass die Daten nur teilweise oder nicht korrekt angezeigt werden, wenn der Scrollbalken am Bildschirmrand bewegt wird.

6.1.1. Die Anwendungsklasse ‚CHexView‘ und die Klasse ‚CMainFrm‘ zur Erstellung des Hauptfensters

Die Klasse ‚CHexView‘ ist die eigentliche Applikationsklasse. Sie wird von der Klasse ‚CWinApp‘ abgeleitet und in ihr wird das eigentliche Applikationsobjekt angelegt.

Im Konstrukt dieser Klasse werden keine Initialisierungen vorgenommen, dies geschieht ausschließlich in der Methode ‚InitInstance‘. Hier werden Schlüsseleinträge in der Registrierung vorgenommen und die im folgenden beschriebenen Klassen ‚CMainFrm‘, ‚CHexViewDoc‘ und ‚CHexViewView‘ in die Applikation integriert. Dies geschieht durch das Anlegen eines neuen SingleDocTemplate-Objektes, welches der Anwendung durch den Aufruf von ‚AddDocTemplate‘ hinzugefügt wird. Am Ende dieser Methode wird noch das Hauptfenster auf dem Bildschirm angezeigt.

In der Klasse ‚CMainFrm‘ werden alle Funktionen zum Aufbau des Hauptfensters zur Verfügung gestellt. So werden hier Statuszeile und Werkzeugleiste angelegt. Außerdem wird das Hauptmenü, welches als Drop-Down-Menü implementiert ist, in das Hauptfenster eingebunden.

6.1.2. Die Klasse ‚CHexViewDoc‘ zur Erstellung eines Dokumentes

Das Programm ‚Datenterminal‘ basiert auf dem sogenannten ‚Single Document Interface‘-Modell (SDI). Bei diesem Verfahren werden die eigentlich Daten von den Ansichten der Daten getrennt. So sind in der Klasse ‚CHexViewDoc‘ nur Methoden implementiert, die direkte Auswirkungen auf die vom Programm ‚Datenterminal‘ zu bearbeitenden Daten haben. Sie ist von der Klasse ‚CDocument‘ abgeleitet, welche schon eine ganze Reihe von Funktionen zur Bearbeitung von Daten zur Verfügung stellt.

Die Methode ‚OnNewDocument‘ dient zum Anlegen eines neuen Dokumentes. Sie löscht eventuell schon im Speicher vorhandene Daten und setzt den Titel des neu angelegten Dokuments auf ‚unbenannt‘. Außerdem wird ein Update der Ansicht der Daten durchgeführt, so dass auch für den Benutzer ersichtlich wird, dass ein neues Dokument angelegt wurde.

Zum Laden von Daten wird die Methode ‚OnOpenDocument‘ verwendet. Ihr wird der Pfad- und Dateiname als Parameter übergeben. Es wird nun versucht, die angegebene Datei zu öffnen. Anschließend wird die Dateilänge bestimmt und ein entsprechend großer Datenpuffer angelegt. Der Inhalt der Datei wird nun in diesen Puffer geladen. Zum Abschluss wird die Datei wieder geschlossen.

Mit der Methode ‚OnSaveDocument‘ können die Daten gespeichert werden. Auch ihr werden Pfad- und Dateiname als Parameter übergeben. Sie versucht, die angegebene Datei zu öffnen, und anschließend die Daten in dieser Datei abzulegen. Auch hier wird die Datei am Ende der Routine wieder geschlossen.

Weitere Methoden dienen zum Zugriff auf die geladenen Daten bzw. auf Eigenschaften der Daten.

Die Methode ‚GetLine‘ liefert die angegebene Anzahl von Datenbytes ab der ebenfalls anzugebenden Position in dem übergebenen Puffer zurück. Sie dient normalerweise dazu, um jeweils 16-Byte-große Ausschnitte der Daten als Zeile auf dem Bildschirm auszugeben. Aus diesem Grund wird hier auch schon die Formatierung der Daten vorgenommen. So wird in dieser Methode die Datenadresse am Anfang der Zeile in hexadezimaler Weise ausgegeben. Auch die 16 Datenbytes werden zunächst hexadezimal dargestellt. Am Ende der Zeile werden alle darstellbaren Zeichen zusätzlich noch als ASCII-Zeichen ausgegeben.

Mit der Methode ‚MakeBuffer‘ wird ein neuer Datenpuffer in der angegebenen Länge angelegt. Ein eventuell schon vorhandener Puffer wird hierbei zunächst gelöscht. Es wird ein Zeiger auf die angelegten Daten zurückgeliefert.

Die Methode ‚KillBuffer‘ dient zum expliziten Löschen des Datenpuffers aus dem Speicher.

Die Methode ‚GetData‘ liefert einen Zeiger auf einen bereits vorhandenen Datenpuffer zurück.

6.1.3. Die Klasse ‚CHexViewView‘ zur Darstellung und Manipulation der Daten

Diese Klasse ist für die Darstellung der Daten zuständig. Sie ist der Kern des Programmes, da in ihr neben der Anzeige der Daten auch Datenmanipulationen, wie das Ent- oder Verschlüsseln, sowie Routinen zum Empfangen und Versenden der Daten bereitgestellt werden.

Die Klasse ‚CHexViewView‘ wird von ‚CScrollView‘ abgeleitet. Damit auch größere Datenmengen angezeigt werden können, wird an der rechten Seite des Fensters ein Scrollbalken dargestellt, so dass der Benutzer auch weiter hinten liegende Daten betrachten kann.

Im Konstruktor dieser Klasse werden eine Reihe von Werten initialisiert, damit sie für die später sofort verwendet werden können. Es werden die Schlüsselbytes alle auf den Wert 0 und die Baudrate für den Empfang und das Versenden von Daten auf 9600 Baud eingestellt, da diese Übertragungsgeschwindigkeit auch im Mikrocontroller-Programm (siehe Kapitel 5.2 – „Die Unterprogramme zur seriellen Übertragung“) verwendet wird. Außerdem wird der COM-Port auf 1 gesetzt und der serielle Port für den Empfang von Daten geöffnet.

In der Methode ‚OnCreate‘, welche automatisch beim Anlegen des HexViewView-Objektes aufgerufen wird, wird zunächst die ‚OnCreate‘-Methode der Vaterklasse aufgerufen. Anschließend wird ein Timer gesetzt, damit in Zeitabständen von circa 2 Sekunden immer wieder geprüft wird, ob Daten an der seriellen Schnittstelle anliegen.

Die eigentliche Ausgabe der Daten erfolgt mit der Methode ‚OnDraw‘. Sie wird immer dann aufgerufen, wenn der Bildschirm neu gezeichnet werden muss, z. B. weil die Größe des Fensters verändert wurde. Das Programm kann die Daten auf zwei Arten ausgeben, und dies sowohl auf dem Bildschirm als auch auf einem Drucker. Daher ist diese Routine in vier Teile aufgeteilt.

Zunächst wird geprüft, ob der Editiermodus aktiv ist. Anschließend wird unterschieden, ob die Daten auf dem Bildschirm oder auf dem Drucker ausgegeben werden sollen.

Im ersten Fall werden die Daten aus dem Editierfeld oder ggf. auch aus dem Speicher in ein CString-Objekt gelesen. Anschließend wird das alte Editierfeld gelöscht und ein neues Editierfeld angelegt, in welches die Daten hineingeschrieben werden. Die Abfrage, ob keine Daten mehr im Speicher vorhanden sind, also ob die Datenlänge 0 ist, wird benötigt, wenn der Benutzer im Editiermodus ein neues Dokument anlegt, damit das Editierfeld gelöscht wird.

Sollen die Daten hingegen auf einem Drucker ausgegeben werden, so wird zunächst ein CRect-Objekt angelegt, welches die Größe eines Blattes hat. Anschließend werden die Daten durchlaufen, bis die Zeile erreicht ist, welche auf dem Blatt als erste ausgegeben werden soll. In einer weiteren Schleife werden dann alle zu druckenden Daten zeilenweise ausgegeben.

Ist der Ansichtsmodus aktiv, so werden die Daten in hexadezimaler Form mit Zeilennummern ausgegeben. Auch hier wird geprüft, ob die Daten auf dem Bildschirm oder auf dem Drucker auszugeben sind.

Im letzteren Fall wird wiederum ein der Blattgröße entsprechendes CRect-Objekt angelegt sowie die erste auf dem aktuellen Blatt auszugebende Zeile bestimmt.

Bei einer Bildschirmausgabe wird die Größe des sogenannten Client-Bereichs des Hauptfensters bestimmt, in welchem die Daten ausgegeben werden sollen. Es wird auch der Scrollbereich der Daten festgelegt und anhand der Position des Scrollbalkens bestimmt, ab welcher Position die Daten ausgegeben werden sollen.

Die Schleife, welche die hexadezimalen Daten wirklich ausgibt, ist für die Bildschirm- und die Druckausgabe identisch: Es werden ab der Startzeile so lange Zeilen aus den Daten extrahiert, bis das Ende des jeweiligen Ausgabe-CRect-Objekts erreicht ist.

Vor dem Aufruf der Methode ‚OnDraw‘ wird z. B. nach dem Laden von Daten oder dem Empfang von Daten an der seriellen Schnittstelle zur Initialisierung die Methode

„OnInitialUpdate“ aufgerufen. Diese Methode stellt die Größe des Scrollbalkens ein. Dies ist allerdings nur erforderlich, wenn das Programm sich nicht im Editiermodus befindet, da dort ohnehin ein Editierfeld mit integriertem Scrollbalken verwendet wird.

Die Methoden, die im folgenden beschrieben werden, wurden aus dem Original-Programm „HexView“ übernommen und lediglich an wenigen Stellen angepasst. Sie dienen dem Ausgeben von Daten auf einem angeschlossenen Drucker. Sie sind kein Bestandteil der eigentlichen Aufgabe, werden hier aber dennoch beschrieben um die Vollständigkeit der Programmdokumentation zu erhalten:

Die Methode „OnPreparePrinting“ bereitet den Druckvorgang vor. Zunächst wird der Standard-Drucker abgefragt. Wenn kein Drucker vorhanden ist, wird durch die MFC-Klassen eine Fehlermeldung generiert und die Routine wird verlassen. Anschließend wird der gewählte Drucker als „belegt“ markiert und der Geräte- und Treibername sowie der Ausgabeport ermittelt. Mit Hilfe dieser Angaben wird ein sogenanntes Device-Context angelegt, welches unter Windows-Betriebssystemen zur Ausgabe verwendet wird. Dieses wird benötigt, um die Anzahl der auszugebenden Seiten berechnen zu können, was durch Aufruf der Methode „CalcPageCount“ geschieht. Anschließend wird das Device-Context wieder gelöscht und die Methode „OnPreparePrinting“ der Vaterklasse (CScrollView) aufgerufen. Diese gibt entweder eine Druckvorschau auf dem Bildschirm aus, ruft den Druckdialog auf oder druckt die Daten direkt aus. Am Ende der Routine „OnPreparePrinting“ wird der belegte Drucker wieder freigegeben.

Mit der Methode „OnBeginPrinting“ wird der Font für die Druckausgabe angelegt. Dazu wird zunächst die Schriftgröße berechnet. Außerdem wird die Seitenanzahl neu berechnet, da der Benutzer ja einen anderen Drucker oder eine andere Blattgröße gewählt haben könnte, so daß sich die Anzahl der Blätter geändert hat.

Die Methode „OnEndPrinting“ bildet das Gegenstück zu der vorher genannten Methode: Sie löscht den angelegten Druckerfont wieder.

Die schon mehrmals erwähnte Methode „CalcPageCount“ ist dafür zuständig die Anzahl der auszugebenden Seiten zu bestimmen. Dazu werden zunächst einmal Breite und Höhe einer Seite, abzüglich des nicht bedruckbaren Randbereiches, ermittelt. Dann muss unterschieden werden, ob das Programm sich im Editiermodus befindet oder nicht, da in den beiden Modi die Anzahl der auszugebenden Zeilen unterschiedlichen ist. Im ersten Fall werden alle Zeilen gezählt, indem der Reihe nach alle Zeilen mit Hilfe der Methode „GetStringLine“ geholt werden, im zweiten wird die Anzahl der Zeilen berechnet, indem die Größe der geladenen Datei durch 16 dividiert werden, da in diesem Fall pro Zeile jeweils 16 Bytes ausgegeben werden. In beiden Fällen wird jedoch anschließend die Anzahl der auszudruckenden Seiten aus der ermittelten Seitengröße und der Zeilenanzahl berechnet. Zusätzlich fließt noch die Schriftgröße in diese Berechnung ein.

Die Methode „OnViewFont“ wird aufgerufen, wenn der Benutzer eine anderen Schriftart wählen möchte. Sie stellt einen Fontauswahldialog dar und legt anschließend den gewählten als neuen Font für die Ausgabe der Daten an.

Mit Hilfe der Methode „MeasureFontHeight“ wird die Schriftgröße ermittelt. Dies geschieht einfach durch die Ausgabe eines beliebigen Textes in ein Dummy-CRect-Objekt. Die Ausgabefunktion „DrawText“ liefert als Rückgabewert die Höhe des ausgegebenen

Textes zurück. Vor dem Aufruf diese Funktion wird noch die vom Benutzer gewählte Schriftart als die aktuelle festgelegt.

Die folgenden Methoden gehören zum Komplex des Versendens und Empfangens von Daten über die serielle Schnittstelle.

In der Methode ‚OnSendSend‘ wird das Dialogfeld, welches zum Versenden der Daten dient, angelegt. Doch bevor es angezeigt wird, werden die zur Zeit angegebenen Schlüsselbytes, die gewählte Baudrate und der gewählte COM-Port in die entsprechenden Elemente des Dialogfeldes eingetragen. Wählt der Benutzer den OK-Button, so werden alle eingetragenen bzw. gewählten Werte nun umgekehrt wieder in die entsprechenden Programmvariablen übernommen. Anschließend wird der Timer, welcher den Empfang von Daten an der seriellen Schnittstelle prüft, gestoppt sowie die entsprechende serielle Schnittstelle geschlossen. Dies ist erforderlich, damit an der gleichen seriellen Schnittstelle Daten versendet und empfangen werden können. Anschließend wird die Methode ‚SendData‘ aufgerufen, welche den eigentlichen Versand der Daten übernimmt. Danach wird der Timer zum Empfang von Daten wieder angelegt und auch die entsprechende serielle Schnittstelle wieder geöffnet.

Die Methode ‚SendData‘ dient zum eigentlichen Versenden der Daten. Zunächst wird hier ein SendDialog-Objekt angelegt, welches den Fortschritt beim Versenden der Daten anzeigt. Anschließend wird versucht, die gewählte serielle Schnittstelle zu öffnen. Ist dies nicht möglich, wird eine entsprechende Meldung für den Benutzer ausgegeben. Dann wird geprüft, ob auch der Schlüssel versendet werden soll. In diesem Fall werden zunächst sechs Schlüsselbytes und die zugehörige Prüfsumme übertragen. Anschließend wird auf ein Acknowledge-Byte gewartet. Wird dieses nicht innerhalb einer Zeit von circa 2 Sekunden gesendet, so wird die Übertragung abgebrochen und eine Meldung an den Benutzer ausgegeben, dass eine Zeitüberschreitung aufgetreten sei. Wird kein ACK- sondern ein NAK-Byte gesendet, so wird der Sendevorgang maximal 100mal wiederholt, danach wird die Übertragung abgebrochen.

Als nächstes wird geprüft, ob überhaupt Daten vorhanden sind, die übertragen werden könnten. Ist dies der Fall, so wird als erstes die Länge der Daten in einem 8-Byte-Block übertragen, von welchen die letzten vier Bytes allerdings den Wert 0 haben. Als neuntes Byte wird auch hier die Prüfsumme übertragen. Nach dem Versand der Daten wird hier ebenfalls auf ein Acknowledge-Byte gewartet, welches wiederum innerhalb der oben genannten Zeit eintreffen muss, damit keine Zeitüberschreitung auftritt. Auch wird die Übertragung, wenn sie erfolglos war, so dass ein NAK-Byte zurückgesendet wurde, maximal 100mal wiederholt, bevor sie wegen der zu schlechten Übertragungsqualität abgebrochen wird.

Vor dem Versand der eigentlichen Daten wird das Dialogfeld, welches den Fortschritt der Übertragung anzeigt, angelegt und der Bereich des Fortschrittbalkens ermittelt. Jetzt folgt die Schleife, in welcher die Daten übertragen werden. Hierzu wird zunächst ein 9-Bytes-großer Puffer angelegt, welche die momentan zu versendenden Daten und im letzten Byte die zugehörige Prüfsumme aufnimmt. Anschließend werden die Daten dann versendet und auf ein Acknowledge-Byte gewartet. Trifft dieses nicht innerhalb eines Zeitintervalls von etwa 2 Sekunden ein, so wird auch hier eine Zeitüberschreitung ausgelöst und der weitere Versand der Daten mit einer Meldung an den Benutzer abgebrochen. Wird statt eines ACK-Byte ein NAK-Byte empfangen, so wird die Übertragung maximal 100mal wiederholt, bevor sie wegen der zu schlechten Übertragungsqualität abgebrochen wird. War die Übertragung

hingegen erfolgreich, so wird der Fortschrittsbalken aktualisiert und die nächsten acht Datenbytes können versendet werden.

Nachdem alle Daten versendet wurden, wird noch eine Endkennung übertragen. Diese besteht, wie in der Protokollbeschreibung bereits beschrieben, aus einer Folge von acht Bytes mit dem Wert ‚0‘ und der Prüfsumme ‚EOT‘, was eigentlich eine Verletzung des Protokolls darstellt. Doch kann genau aus diesem Grund das Ende der Übertragung erkannt werden. Nachdem Versenden dieser Kennung wird ebenfalls auf ein Acknowledge-Byte gewartet und die Übertragung wiederum maximal 100mal wiederholt, wenn ein NAK-Byte empfangen wurden. Wie bei den zuvor beschriebenen Übertragungen wird eine Zeitüberschreitung ausgelöst, wenn dieses Byte nicht innerhalb eines Zeitintervalls von 2 Sekunden empfangen werden konnte.

In der Methode ‚OnSendReceiveproperties‘ wird ein Dialogfeld aufgebaut, welches es dem Benutzer ermöglicht, Einstellungen für den Empfang von Daten vorzunehmen. Dies sind die Übertragungsgeschwindigkeit und der serielle Port, an welchem die Daten empfangen werden sollen. Bevor dieses Dialogfeld angezeigt wird, werden die zuvor gewählten Werte eingestellt. Hat der Benutzer seine Einstellungen beendet und wählt er den OK-Button, so werden die ausgewählten Einstellungen ausgewertet und die entsprechenden globalen Variablen gesetzt. Außerdem wird der COM-Port zum Empfangen der Daten auf die neuen Werte eingestellt.

Der eigentlich Empfang der Daten findet durch die Methode ‚OnTimer‘ statt. Diese Methode wird automatisch durch einen nach dem Programmstart gesetzten Timer in Zeitabständen von circa 2 Sekunden aufgerufen. Es wird daher zunächst geprüft, ob überhaupt Daten an dem zum Empfang gewählten seriellen Port anliegen. Ist dies nicht der Fall, wird die Routine sofort wieder verlassen. Im anderen Fall wurde jedoch von der Gegenseite eine Übertragung gestartet.

Dann wird geprüft, ob bereits der erste 9-Byte-Block vollständig empfangen wurde. Es wird erwartet, dass die Gegenseite das verwendete Protokoll einhält und immer in 9-Byte-Blöcken überträgt. Können keine 9 Bytes empfangen werden, so wird die Übertragung nach einer gewissen Zeit aufgrund einer Zeitüberschreitung abgebrochen. Dieser Block wird geprüft, indem eine Prüfsumme über die ersten acht Bytes berechnet wird und diese mit dem neunten Byte verglichen wird. Sind diese identisch, wird ein ACK-Byte als Acknowledge zurückgesendet, ansonsten ein NAK-Byte, damit der Empfänger die Übertragung wiederholt. Dies kann maximal 100mal geschehen, dann wird die Übertragung aufgrund der zu schlechten Übertragungsqualität abgebrochen.

Wurde jedoch der erste 9-Byte-Block komplett und korrekt empfangen, so handelt es sich hierbei um die Längeninformation, aus der diese Routine herauslesen kann, wie viele Daten im Verlauf der weiteren Übertragung empfangen werden sollen und wie groß der anzulegende Puffer sein muss. Außerdem wird dieser Wert dazu verwendet, um dem Benutzer in einem Dialogfeld den Fortschritt des Empfanges der Daten anzeigen zu können. Dieses Dialogfeld wird zwar schon zu Beginn der Routine angelegt, aber erst nach dem erfolgreichen Empfang der Länge angezeigt.

Der Empfang der eigentlichen Daten verläuft analog zum Versenden derselben, d. h. es wird so lange gewartet, bis ein 9-Byte-Block komplett empfangen wurde. Dauert dies zu lange, wird die Übertragung wegen Zeitüberschreitung abgebrochen. Ansonsten wird geprüft, ob die Prüfsumme korrekt war. Ist dies der Fall, so wird als Acknowledge ein ACK-Byte

zurückgesendet, ansonsten ein NAK-Byte, um dem Sender zu signalisieren, dass er die Übertragung des letzten Blocks wiederholen soll. Auch hier kann dies maximal 100mal geschehen, bevor die Übertragung wegen zu schlechter Qualität abgebrochen wird. Beim Überprüfen muss allerdings beachtet werden, dass es sich ja auch um die Endkennung handeln kann, bei welcher die empfangene mit der berechneten Prüfsumme nicht übereinstimmt. Dieser Fall muss bei der Überprüfung erkannt werden.

Bevor der nächste Datenblock empfangen wird, wird noch der Dialog, in welchem der Fortschritt der Übertragung angezeigt wird, erneuert. Anschließend wird gewartet, ob ein weiterer 9-Byte-Block empfangen werden kann. Ist dies der Fall, so werden die bereits genannten Schritte erneut durchgeführt. Ansonsten wird angenommen, dass die Übertragung beendet ist.

Die folgenden Methoden dienen der Ver- bzw. Entschlüsselung der Daten. Die Implementation dieser Routinen auch in das Programm zum Versenden bzw. Empfangen der Daten hat den Vorteil, dass es nicht unbedingt zweier Leitungsverchlüsseler bedarf, sondern auch verschlüsselte Daten direkt empfangen und dann vom Empfangsprogramm entschlüsselt werden können.

Beim Aufruf der Methode ‚OnEncrypt‘ werden zunächst die aktuell verwendeten Schlüsselbytes in die Editierfelder eines Dialogfeldes geschrieben, in welchem der Benutzer entscheiden kann, welchen Verschlüsselungsalgorithmus er anwenden möchte, und einen entsprechenden Schlüssel eingeben kann. Als Verschlüsselungsalgorithmus steht dem Benutzer neben der einfachen Verschlüsselung, welche die Daten nach dem auch vom Mikrocontroller implementierten FC1-Algorithmus chiffriert, auch die Möglichkeit zur Verfügung, die Daten mittels des bekannten RC5-64-Algorithmus zu verschlüsseln.

Nach Anwählen des OK-Buttons werden die in den Editierfeldern zur Schlüsseleingabe enthaltenen Werte zunächst geprüft und bei Korrektheit in die entsprechenden Programmvariablen übernommen. Anschließend wird geprüft, welche Verschlüsselungsmethode der Benutzer gewählt hat und der entsprechende Verschlüsselungsalgorithmus wird gestartet. Diese Algorithmen sind an dieser Stelle direkt implementiert.

Die Methode ‚OnDecrypt‘ ist analog zur Methode ‚OnEncrypt‘ aufgebaut. Sie dient lediglich dem Entschlüsseln der Daten nach Eingabe eines korrekten Schlüssels, so dass an dieser Stelle die statt der Ver- die entsprechenden Entschlüsselungsalgorithmen implementiert sind.

Die Methode ‚OnBruteForce‘ bietet dem Benutzer die Möglichkeit, einen verschlüsselten Text auch ohne Kenntnis des korrekten Schlüssels zu entschlüsseln, indem einfach der Reihe nach alle möglichen Schlüssel durchprobiert werden. Hierfür ist allerdings die Kenntnis eines Teils der Nachricht in unverschlüsselter Form notwendig, welchen der Benutzer eingeben muss, damit nach dem Aufruf des Verschlüsselungsalgorithmus automatisch geprüft werden kann, ob der gesuchte Schlüssel gefunden wurde. Die Methode durchsucht einfach den durch die vorangegangene Entschlüsselung gewonnenen Text nach dem vom Benutzer eingegebenen Klartext (siehe Kapitel 2.4. – *„Ein Brute-Force-Angriff auf den FC1-Algorithmus“*).

In der Methode ‚OnViewFont‘ wird ein Fontauswahl-Dialogfeld dargestellt. Betätigt der Benutzer den OK-Button, so wird versucht, den ausgewählten Font zur Darstellung der Daten zu verwenden.

In der Methode ‚OnViewEditfield‘ kann zwischen dem Editiermodus und dem Modus, welcher die Daten in hexadezimaler Form darstellt, umgeschaltet werden. Dazu wird die Boolean-Variable ‚showedit‘ geprüft. Ist sie gesetzt, so soll der Editiermodus aktiv sein und es wird zunächst die Methode ‚OnInitialUpdate‘ aufgerufen, um einen eventuell schon vorhandenen Scrollbalken des Hauptfensters zu entfernen. Anschließend wird ein Editierfeld angelegt, welches den gesamten Client-Bereich des Hauptfensters einnimmt. Nun werden bereits geladene Daten als Inhalt in dieses Editierfeld geschrieben.

Soll hingegen vom Editiermodus zum Darstellungsmodus umgeschaltet werden, so wird zunächst geprüft, ob die Daten im Editiermodus verändert wurden. In diesem Fall muss der Datenpuffer komplett erneuert werden und der Inhalt des Editierfeldes wird in den neuen Puffer kopiert. Außerdem wird das ‚ModifiedFlag‘ des Dokumentes gesetzt um anzuzeigen, dass das Dokument verändert wurde. Anschließend wird das Editierfeld entfernt und gelöscht sowie die Methode ‚OnInitialUpdate‘ aufgerufen, um auch die Bildschirmausgabe in den Darstellungsmodus umzuschalten.

Die Methoden, welche mit dem Präfix ‚OnUpdate‘ beginnen, beziehen sich alle auf die Menüpunkte des Hauptmenüs. In ihnen werden je nach Status des Programmes die Menüpunkte aktiviert oder deaktiviert, also grau dargestellt, so dass sie nicht angewählt werden können. Bei dem Menüpunkt ‚Editierfeld‘ wird in der ‚OnUpdateViewEditfield‘-Methode hingegen ein Haken vor den Menüpunkt gesetzt bzw. dieser wird entfernt um anzuzeigen, welcher Modus aktiv ist.

6.1.4. Die Klasse ‚CSerial‘ zur Kommunikation über die serielle Schnittstelle

Die Klasse ‚CSerial‘ wurde ebenfalls [LeAr1998] entnommen. Sie stellt die Methoden zur Kommunikation über die serielle Schnittstelle bereit.

Im Konstruktor werden Initialisierungen durchgeführt, d. h. einige Variablen auf Vorgabewerte gesetzt, im Destruktor wird der serielle Kommunikationskanal zur Sicherheit noch geschlossen, falls das nicht im aufrufenden Programm geschieht.

Die Methode ‚Open‘ dient zum Öffnen einer seriellen Schnittstelle. Hier wird die Nummer des COM-Ports, über welchen kommuniziert werden soll, sowie die Datenübertragungsrate (Baudrate) festgelegt, welche dieser Methode als Parameter übergeben werden müssen. Konnte der serielle Kommunikationskanal erfolgreich geöffnet werden, so liefert diese Methode ein ‚true‘, ansonsten den Wert ‚false‘. Nach dem erfolgreichen Öffnen wird auch die interne boolesche Variable ‚m_bOpened‘ auf den Wert ‚true‘ gesetzt. Auch ist in diesem Fall ein Handle auf die Datei, welche den seriellen Port repräsentiert, entsprechend gesetzt.

Mit der Methode ‚Close‘ kann der serielle Kommunikationskanal wieder geschlossen werden. Hier wird die genannte Variable ‚m_bOpened‘ wieder auf ‚false‘ sowie das Dateihandle zurückgesetzt.

Mit Hilfe der Methode ‚SendData‘ kann ein Datenpuffer, welcher in Form eines Zeigers auf ein Charakter-Feld übergeben wird, über die serielle Schnittstelle versendet werden. Zusätzlich muss auch noch die Anzahl der Bytes, die versendet werden sollen, übergeben werden. Die Methode durchläuft dann den übergebenen Puffer, und versendet so lange einzelne Bytes mittels der nur innerhalb des Serial-Objekts zugänglichen Methode ‚WriteCommByte‘, bis die angegebene Anzahl übermittelt wurde. Zuvor wird allerdings

geprüft, ob die serielle Schnittstelle zuvor überhaupt geöffnet wurde und ob auch ein entsprechendes Dateihandle existiert. Ist eines der beiden genannten Punkte der Fall, so wird die Methode sofort verlassen und liefert als Rückgabewert ‚0‘. Wenn jedoch Daten versendet werden konnten, so liefert die Methode die Anzahl der übermittelten Datenbytes.

Die Methode ‚ReadDataWaiting‘ wird eingesetzt um zu überprüfen, ob Daten zum Empfang an der seriellen Schnittstelle bereitstehen. Ist dies der Fall, so liefert die Methode die Anzahl der zu empfangenden Bytes als Rückgabewert.

Mit der Methode ‚ReadData‘ kann schließlich die im Parameter ‚limit‘ übergebene Anzahl von Bytes in den ebenfalls übergebenen Puffer gelesen werden. Sie liefert als Rückgabewert die Anzahl der tatsächlich gelesenen Datenbytes.

6.1.5. Dialogklassen zur Benutzerkommunikation

Die Klassen ‚BruteForce‘, ‚CenterKeyDialog‘, ‚CReceive‘, ‚CSendDialog‘ und ‚CSending‘ dienen alle zur Darstellung jeweils eines Dialogfelds um mit dem Benutzer zu kommunizieren. Sie sind daher alle von der Klasse ‚CDialog‘ abgeleitet und haben einen nahezu identischen Aufbau.

Im Konstruktor werden die sogenannten Member-Variablen, welche den Zustand oder den Inhalt der Kontrollelemente des Dialogs repräsentieren, mit Vorgabewerten initialisiert.

Die Methode ‚DoDataExchange‘ dient dazu, die Inhalte oder Zustände der einzelnen Kontrollelemente verifizieren und in die entsprechenden Variablen kopieren. Die Überprüfung des Inhalts wird durch Funktionen mit dem Präfix ‚DDV_‘ (DialogDataVerification) durchgeführt, während das Kopieren der Daten durch Methoden mit dem Präfix ‚DDX_‘ (DialogDataExchange) geschieht. In den Klassen ‚CenterKeyDialog‘ und ‚CSendDialog‘ ist für die Verification der Daten die Methode ‚DDV_MyCustom‘ implementiert, welche dazu dient zu überprüfen, ob die beiden zusammengehörenden Schlüsselbytes von Schlüssel und Bestätigung jeweils identisch sind. Ist dies nicht der Fall, so erfolgt eine entsprechende Meldung an den Benutzer und der Eingabecursor wird in das Eingabefeld der Bestätigung gesetzt, in welchem der Unterschied festgestellt wurde.

In einigen dieser Klassen wird zusätzlich noch eine sogenannte *Message Map* angelegt, um sofort innerhalb der Dialogklasse auf Benutzeraktionen zu reagieren. So werden in allen Dialogen, welche die Eingabe von Schlüsselbytes ermöglichen, die Eingabefelder erst aktiviert, wenn eine Checkbox oder der entsprechende Radiobutton gewählt wurde. Aus diesem Grund existiert in diesen Dialogen auch eine Neudefinition der Methode ‚OnPaint‘, damit beim Anlegen des Dialogs die entsprechenden Eingabefelder sofort korrekt dargestellt, also je nach dem aktiviert oder deaktiviert werden. Des weiteren sind in diesen Dialogen auch weitere Methoden mit dem Präfix ‚On‘ vorhanden, welche in der MessageMap den einzelnen Ereignissen (Events) zugeordnet werden, um die Schlüsseleingabefelder sofort beim Anwahl des entsprechenden Radiobuttons bzw. der Checkbox zu aktivieren oder zu deaktivieren.

6.1.6. Die Klasse ‚KeyManagement‘ zur Verwaltung der Schlüssel

Hierbei handelt es sich ebenfalls um eine von CDialog abgeleitete Klasse. In ihr können die verwendeten Schlüssel verwaltet werden.

Im Konstruktor der Klasse werden die Elemente des Dialogfeldes initialisiert. So werden die Schlüsselwerte alle auf ‚0‘ gesetzt und das Eingabefeld für das Schlüsselwort gelöscht. Außerdem wird der Pfad- und Dateiname der Datei festgelegt, in welcher der Schlüssel, das zugehörige Schlüsselwort sowie Datum und Uhrzeit für den jeweiligen Schlüssel gespeichert werden. Zu diesem Zweck wird der Pfad- und Dateiname der Hilfedatei ausgelesen und an diesen Namen die Endung ‚.dat‘ an die Stelle der Endung ‚.hlp‘ geschrieben. Dies ist erforderlich, damit die Schlüssel für alle verschlüsselten Dateien in der gleichen Datei gespeichert werden, auch wenn die verschiedenen Dateien, mit welchen der Benutzer arbeitet, in unterschiedlichen Verzeichnissen gespeichert sind.

Die Methode ‚DoDataExchange‘ wird immer dann aufgerufen, wenn die eingegebenen Daten überprüft werden müssen. So wird hier sichergestellt, dass für Datum und Uhrzeit nur gültige Werte eingetragen werden. Außerdem liest diese Methode die Editierfelder des Dialogs aus und schreibt die gefundenen Werte in die zugehörigen lokalen Variablen.

In der Methode ‚OnSave‘ werden die eingegebenen Daten in der oben erwähnten Datei gespeichert. Hierzu wird zunächst ein Objekt der Klasse ‚CFile‘ angelegt. Mittels der Funktion ‚UpdateData‘ werden die in die Editierfelder eingegebenen Daten in die lokalen Variablen gelesen. Anschließend werden 35 Bytes an Speicherplatz reserviert, welcher zum Speichern eines einzelnen Dateieintrages benötigt wird, der eine Länge von genau 35 Bytes hat. Danach werden die Werte der lokalen Variablen in diesen Speicherplatz geschrieben, zunächst die einzelnen Schlüsselbytes, dann die Bytes des Schlüsselwortes und zuletzt noch die Bytes, welche zum Speichern des Datums und der Uhrzeit benötigt werden. Nun wird versucht, die Datei zu öffnen und den gewünschten Eintrag an das Ende dieser Datei zu schreiben. Treten hierbei Fehler auf, so wird der Benutzer benachrichtigt. Am Ende dieser Methode wird der für den Dateieintrag benötigte Speicherplatz wieder freigegeben.

Die auf diese Weise angelegte und mit Daten befüllte Datei kann mit der Methode ‚OnSearch‘ wieder ausgelesen werden. Da beim Durchsuchen der Datei das eingegebene mit dem in der Datei gefundenen Schlüsselwort verglichen werden, wird zunächst geprüft, ob dieses überhaupt eingegeben wurde. Ist dies nicht der Fall, so wird der Benutzer aufgefordert, ein Schlüsselwort einzugeben. Auch hier werden zunächst ein Objekt der Klasse ‚CFile‘ sowie Speicherplatz für einen Dateieintrag angelegt. Außerdem werden noch ein Objekt vom Typ ‚CString‘ für den Vergleich des eingegebenen und des in der Datei gefundenen Schlüsselwortes und ein Objekt vom Typ ‚COleDateTime‘ für das gefundene Datum sowie eine Zählervariable angelegt. Mit Hilfe einer Variable des Typs ‚bool‘ kann entschieden werden, ob die Schleife, mit welcher die Datei durchsucht wird, verlassen werden kann. Im Anschluß wird die Datei zum Lesen geöffnet und der Dateizeiger an den Anfang der Datei gesetzt. In der folgenden Schleife wird bei jedem Durchlauf ein 35-Byte-großer Dateieintrag ausgelesen. Dieser Eintrag wird in seine einzelnen Bestandteile, also Schlüsselbytes, Schlüsselwort sowie Datum und Uhrzeit zerlegt und in die entsprechenden Variablen geschrieben. Vor Ende der Schleife wird dann überprüft, ob das eingegebene mit dem gelesenen Schlüsselwort übereinstimmt. Ist dies der Fall, so wird die boolsche Variable auf den Wert ‚true‘ gesetzt, so dass die Schleife verlassen werden kann. Ansonsten muss der nächste Dateieintrag gelesen werden. Wurde das Ende der Datei erreicht, ohne dass das gesuchte Schlüsselwort gefunden werden konnte, so wird die Schleife ebenfalls verlassen, allerdings hat jetzt die boolsche Variable immer noch den Wert ‚false‘. Auf diese Weise kann dann nach der Schleife festgestellt werden, ob der gesuchte Schlüssel gefunden wurde, so dass die gelesenen Schlüsselbytes sowie Datum und Uhrzeit in die lokalen Variablen übernommen werden können. Ansonsten wird dem Benutzer mitgeteilt, dass der gesuchte

Schlüssel nicht gefunden werden konnte. Auch hier wird am Ende der Methode benötigter Speicherplatz wieder freigegeben.

6.1.7. Das Klassenmodell

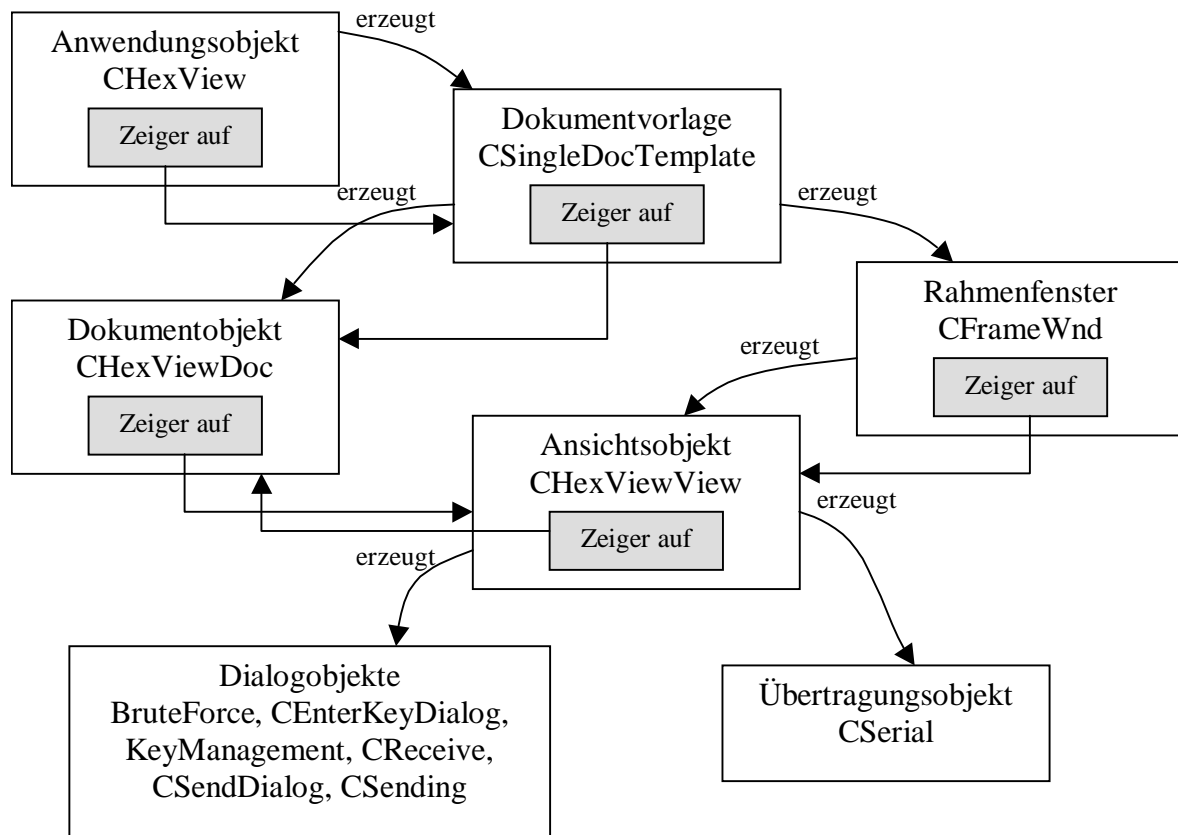


Abb. 11 - Das Klassenmodell (aus [LeAr1998])

6.2. Fehler oder Fehlfunktionen im Programm ‚Datenterminal‘

Trotz umfangreicher Tests und Untersuchungen des Programmcodes ließen sich verschiedene Fehler oder Fehlfunktionen nicht beheben.

6.2.1. Fehler im Anzeigemodus

Es bereitet dem Programm Probleme, große Dateien korrekt anzuzeigen. Es wird zwar die komplette Datei angezeigt, allerdings läßt sich bei Dateien, die eine Größe von circa 26 Kilobyte überschreiten, der Laufbalken nicht mehr mit der Maus direkt positionieren, wenn die normale Systemschriftart (Fixedsys) und –größe (11 Punkt) verwendet wird. Wird bei einer solchen Datei der Laufbalken manuell mit der Maus und gedrückter Maustaste zu weit nach unten positioniert, so springt der Laufbalken nach dem Loslassen der Maustaste wieder nach oben zurück. Lediglich durch Klicken auf die Lauffläche des Scrollbalkens oder durch Anklicken der Pfeile kann das Ende der Datei eingesehen werden. Allerdings wird auch hierbei die fehlerhafte Anzeige sichtbar, da an bestimmten Stellen keine Daten angezeigt werden. Die Anzahl dieser Stellen variiert je nach Größe der Datei. Wird eine geringere Schriftgröße (und ggf. auch ein andere Schriftart) verwendet, so tritt dieser Fehler erst bei größeren Dateien auf, bei größerer Schrift jedoch schon bei kleineren Dateien.

Eine genauere Untersuchung dieser Problematik ergibt, dass es sich hierbei um einen Fehler in der Windows-Funktion handelt, welche dazu dient um die Größe des Scrollbalkens anzupassen. Wenn nur die schon in dem Programm ‚HexView‘ verwendete Funktion

„SetScrollSizes“ verwendet wird, die in der Methode „OnInitialUpdate“ der Klasse „CHexViewView“ aufgerufen wird, wird von einer zu großen geladenen Datei nur der erste Teil angezeigt. Wird zusätzlich in der Methode „OnDraw“ noch die Funktion „SetScrollRange“ aufgerufen, so kann zwar die gesamte Datei angezeigt werden, jedoch nur mit den oben beschriebenen Fehlern. Es zeigt sich, dass dieses Phänomen vermutlich auf einen Überlauf zurückzuführen ist, da zu große Dateien bei Verwendung nur der Methode „SetScrollSizes“ je nach Größe bis zu unterschiedlichen Stellen angezeigt werden und nicht bis zu einer bestimmten Stelle. Außerdem kann der MSDN-Hilfe zu Visual C++ [Mi1998] entnommen werden, dass die Differenz zwischen Anfangs- und Endwert eines Scrollbalkens maximal 32767 betragen darf, was der maximale Wert für eine vorzeichenbehaftete 16-Bit-Integer-Zahl ist. Da bei Dateien ab einer Größe von ca. 26 KB dieser Wert z. T. deutlich überschritten wird, kommt es zu den oben beschriebenen Fehlern.

6.2.2. Fehler im Editiermodus

Ein weiterer Fehler zeigt sich, wenn bei Nicht-Text-Daten in den Editiermodus gewechselt wird. Da diese Daten nicht als Text angezeigt werden können, ist meist nur ein Teil der Daten zu sehen. Einige Bytes werden als Sonderzeichen angezeigt, andere können überhaupt nicht dargestellt werden. Wird anschließend in den Anzeigemodus zurückgewechselt, so fehlt meist ein Teil der ursprünglichen Daten. Ein Grund hierfür kann in der Eigenschaft von C-Strings gesehen werden, bei welchen ein Zeichen mit dem ASCII-Code 0 als Endkennung angesehen wird. Enthalten die Daten also an irgendeiner Stelle den Wert 0, so werden die Daten an dieser Stelle beim Wechsel in den Editiermodus abgeschnitten, da die Daten zur Darstellung in einem Editierfeld in einen solchen C-String konvertiert werden müssen. Weil das Programm aber nicht automatisch erkennen kann, ob es sich bei den Daten um reinen Text handelt oder nicht, kann der Editiermodus auch nicht für derartige Daten einfach gesperrt werden.

Der Benutzer wird jedoch beim Wechsel in den Editiermodus darauf hingewiesen, dass es sich möglicherweise nicht um reine Textdaten handelt, wenn innerhalb der Daten eine binäre Null gefunden wird. Er kann dann entscheiden, ob er die Daten wirklich editieren möchte oder nicht.

7. Diskussion

Mit dem vorliegenden Projekt konnte gezeigt werden, dass es selbst mit relativ einfachen Mitteln möglich ist, einen Schutz bei der Übertragung von Daten zu implementieren. Leider war es jedoch nicht möglich, bekanntere Verschlüsselungsalgorithmen wie RC5 oder DES auf dem für das Projekt genutzten 8-Bit-Mikrocontroller zu implementieren.

Gründe hierfür können zum einen in der Hardware des Mikrocontrollers gesehen werden, welcher lediglich über 8-Bit-Register verfügt und daher eine 16- oder gar 32-Bit-Arithmetik, die für derartige Verschlüsselungsalgorithmen benötigt wird, sehr erschwert. Zum anderen war es für die Datenübertragung unumgänglich, den Mikrocontroller in Assembler zu programmieren, so dass auch kein C-Compiler genutzt werden konnte, der 16- oder 32-Bit-Operationen für einen Mikrocontroller zur Verfügung stellt.

Da die Möglichkeiten der Mikrocontrollerschaltung relativ begrenzt sind, wurde in dem vorliegenden Projekt besonderen Wert auf die Implementierung der Software auf dem Datenendgerät gelegt, in dem vorliegenden Fall also ein Personal Computer. Die Beschreibung dieser Software stellte einen großen Teil der Arbeit dar und wird daher in der vorliegenden Dokumentation auch ausführlich beschrieben. Derartige Software wird sehr in vielen Fällen unter großem Zeitdruck hergestellt, so dass die Dokumentation oft darunter leidet. Es wurde daher versucht, diese so ausführlich wie möglich zu gestalten, um auch einzelne Besonderheiten, die bei der Implementierung genutzt wurden, nachvollziehen zu können.

Dieses Projekt könnte dahingehend erweitert werden, dass die eingesetzte Schaltung zwischen ein Modem und die serielle Schnittstelle eines Datenendgerätes gesetzt werden könnte, um auch die über das öffentliche Telefonnetz laufende Daten verschlüsseln zu können. Zum einen müsste die Schaltung hierzu jedoch alle Modembefehle sowie deren Länge erkennen, da diese nicht verschlüsselt werden dürfen. Außerdem muss vor dem Beginn der Übertragung ein aktueller Schlüssel in den Leitungsverchlüsseler geladen werden. Derartige Funktionalität bringt eine sehr starke Veränderung des verwendeten Protokolls mit sich und würde daher den Rahmen dieser Studienarbeit sprengen.

Trotzdem konnte gezeigt werden, welche Arten von Verschlüsselung selbst mit einfachen Mitteln möglich sind. Auch eine weitere Verwendung dieser Studienarbeit in späteren Projekten sollte problemlos möglich sein.

Anhang

A - Die Bedienung des Programmes „*Datenterminal*“

Die Bedienung des Programmes „*Datenterminal*“ erfolgt entsprechend der eines GUI-Windows-Programmes. Trotzdem soll die einzelnen Menüpunkte und Benutzereingaben im folgenden näher erläutert werden:

a) Das Menü ‚*Datei*‘

Der Menüpunkt ‚*Neu*‘ aus dem Menü ‚*Datei*‘ dient dazu ein neues, leeres Dokument anzulegen. Bereits vorhandene Daten werden nach Anwahl dieses Menüpunktes gelöscht.

Durch Anwählen des Menüpunktes ‚*Öffnen*‘ können bereits vorhandene Dokumente geöffnet werden. Es öffnet sich ein Dateiauswahldialog, in welchem die zu öffnende Datei gewählt werden kann. Als Dateitypen stehen entweder Textdateien (*.txt) oder alle Arten von Dateien (*.*) zur Verfügung.

Mit dem Menüpunkt ‚*Sichern*‘ kann das gerade geöffnete Dokument unter dem bereits vorhandenen Namen gesichert werden. Wenn ein neues Dokument angelegt wurde, welches noch keinen Namen („*unbenannt*“) besitzt, so wird hier ebenfalls ein Dateiauswahldialog angezeigt, in welchem ein Namen für das zu sichernde Dokument anzugeben ist.

Der Menüpunkt ‚*Sichern unter*‘ dient dazu, das geöffnete Dokument unter einem neuen Namen zu sichern. Beide Menüpunkte zum Speichern von Dokumenten sind im Editiermodus deaktiviert. Hier muss zunächst in den Anzeigemodus gewechselt werden, bevor das Dokument gesichert werden kann.

Das Dokument kann durch Anwahl des Menüpunktes ‚*Drucken*‘ auf einem beliebigen, innerhalb des aktiven Windows-Systems eingerichteten Drucker ausgegeben werden. Nach Anwahl dieses Menüs erscheint zunächst noch ein Druckdialog, in welchem verschiedene Einstellungen (z. B. Anzahl der Kopien, zu druckende Seiten usw.) durchgeführt werden können. Ist der Editiermodus aktiv, so werden die Daten als ASCII-Zeichen gedruckt, wie sie auch auf dem Bildschirm angezeigt werden. Im Anzeigemodus werden sie ebenfalls in der Form gedruckt, in welcher sie auf dem Bildschirm dargestellt werden.

Der Menüpunkt ‚*Druckvorschau*‘ dient dazu, sich schon vor dem eigentlichen Drucken ein Bild davon machen zu können, wie die Daten auf dem Papier ausgegeben werden. Auch hier werden die Daten entsprechend des aktiven Modus (Anzeige- oder Editiermodus) angezeigt. Die Blätter können hier in drei verschiedenen Vergrößerungsstufen betrachtet werden. Außerdem können sie auch direkt ausgedruckt werden, ohne die Druckschau verlassen und den Menüpunkt ‚*Drucken*‘ wählen zu müssen.

Nach Anwählen des Menüpunktes ‚*Druckereinrichtung*‘ erscheint ein Dialogfeld, in welchem der Benutzer den gewünschten Drucker, sowie dessen Eigenschaften (Papierformat, Hoch- oder Querformat usw.) festlegen kann.

Unter diesem Menüpunkt werden nach einer Trennlinie die vier zuletzt geöffneten Dokumente angezeigt, welche durch direktes Anwählen erneut geöffnet werden können.

Der Menüpunkt ‚*Beenden*‘ beendet das Programm. Haben sich die Daten seit der letzten Speicherung verändert, so wird der Benutzer gefragt, ob er diese Änderungen vor dem Beenden noch sichern möchte.

b) Das Menü ‚*Bearbeiten*‘

Die Menüpunkte dieses Menüs sind nur im Editiermodus aktiviert. Sie dienen zum Datenaustausch mit der Zwischenablage und bieten die dort bekannten Funktionen ‚*Rückgängig*‘, ‚*Ausschneiden*‘, ‚*Kopieren*‘ und ‚*Einfügen*‘, welche die benannten Funktionen innerhalb des Editierfeldes ausführen.

c) Das Menü ‚*Ansicht*‘

In diesem Menü können verschiedene Ansichtseigenschaften eingestellt werden:

Die beiden Menüpunkte ‚*Werkzeugleiste*‘ und ‚*Statuszeile*‘ dienen dazu, die entsprechenden Elemente des Hauptfensters ein- oder auszublenden. Je nachdem, ob diese Elemente ein- oder ausgeblendet wurden, wird ein Haken vor dem entsprechenden Menüpunkt angezeigt.

Mit dem Menüpunkt ‚*Schriftart*‘ wird ein Auswahldialog aufgerufen, in welchem eine Schriftart für die Ausgabe gewählt werden kann. Im Editiermodus ist allerdings darauf zu achten, dass hier der gewählte Font lediglich Auswirkungen auf die Druckausgabe, nicht jedoch auf die Bildschirmausgabe hat. Im Anzeigemodus wird hingegen für beide Ausgabeformen die hier gewählte Schriftart genutzt.

Zwischen Editier- und Anzeigemodus kann durch Anwählen des Menüpunktes ‚*Daten editieren*‘ gewechselt werden. Im Editiermodus wird vor diesem Menüpunkt daher ein Haken angezeigt.

d) Das Menü ‚*Senden*‘

Unter diesem Menü können die Einstellungen zum Versenden und zum Empfangen der Daten vorgenommen werden.

Wird der Menüpunkt ‚*Daten versenden*‘ gewählt, so erscheint Dialogfeld, in welchem der COM-Port, über den die Daten versendet werden sollen, sowie die Übertragungsgeschwindigkeit eingestellt werden können. Wenn die Checkbox ‚*Schlüssel versenden*‘ gewählt wurde, so können in den darunterliegenden Editierfeldern die sechs am Beginn der Übertragung zu versendenden Schlüsselbytes eingegeben werden. Hierbei sind nur Werte zwischen 0 und 255 zulässig. Im Editiermodus können keine Daten versendet werden, so dass dieser Menüpunkt dann deaktiviert ist. Ist kein Dokument geladen, so ist dieser Menüpunkt ebenfalls deaktiviert. Wird der ‚*OK*‘-Button gewählt, so werden die Daten mit den vorgenommenen Einstellungen versendet.

Mit dem Menüpunkt ‚*Empfangseigenschaften*‘ lassen sich Einstellungen zum Empfang der Daten vornehmen. Hier müssen lediglich der COM-Port und die Übertragungsgeschwindigkeit festgelegt werden. Diese Werte werden allerdings erst nach dem Betätigen des ‚*OK*‘-Buttons aktiv.

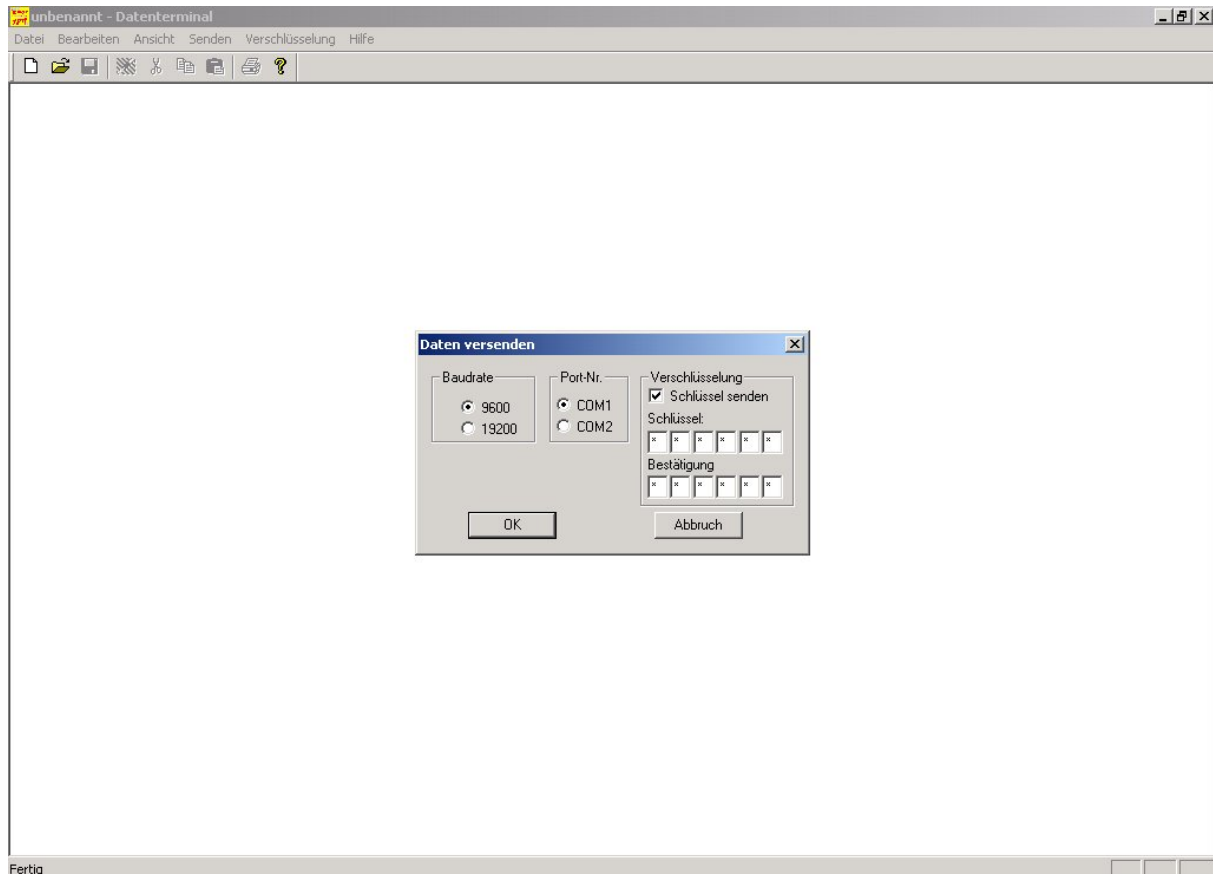


Abb. 12 - Das Datenterminal-Programm mit Dialog zum Versenden

e) Das Menü ‚Verschlüsselung‘

Die Menüpunkte innerhalb des Menüs ‚Verschlüsselung‘ sind ebenfalls nur im Anzeigemodus aktiv und auch nur dann, wenn tatsächlich Daten vorhanden sind.

Nach Anwahl des Menüpunktes ‚Verschlüsseln‘ erscheint ein Dialogfeld, in welchem der Benutzer mit Hilfe zweier Radiobuttons entscheiden kann, nach welchem Algorithmus die Daten verschlüsselt werden sollen. Der Punkt ‚Einfache Verschlüsselung‘ bedeutet, dass die Daten nach dem gleichen Algorithmus zu verschlüsseln sind, welcher auch von dem angeschlossenen Leitungsverschlüsseler verwendet wird. Hierbei sind lediglich sechs Schlüsselbytes anzugeben. Wird hingegen der Punkt ‚RC5-64 Verschlüsselung‘ gewählt, so wird zu Verschlüsselung der RC5-Algorithmus mit einer Schlüssellänge von 64 Bit verwendet, so dass insgesamt 8 Schlüsselbytes angegeben werden müssen und auch noch die beiden hinteren Editierfelder aktiviert werden. Nach dem Anwählen des Buttons ‚OK‘ werden die Daten dann verschlüsselt.

Wählt der Benutzer den Menüpunkt ‚Entschlüsseln‘, so erscheint das gleiche Dialogfeld, was auch zum Verschlüsseln der Daten verwendet wird. Es können hier also die gleichen Einstellungen vorgenommen werden. Nur wird hier nach Klicken des ‚OK‘-Buttons nicht der gewählte Ver- sondern der entsprechende Entschlüsselungsalgorithmus verwendet.

Der Menüpunkt ‚Brute force Angriff‘ gibt dem Benutzer die Möglichkeit, einen mittels einfacher Verschlüsselung verschlüsselten Geheimtext, der auch vom Leitungsverschlüsseler erzeugt worden sein kann, auch ohne Kenntnis des Schlüssels zu entschlüsseln. Hierzu muss ihm allerdings der unverschlüsselte Anfang des Textes bekannt sein. Hat der Benutzer diesen

in dem erscheinenden Dialogfeld eingegeben und den ‚OK‘-Button betätigt, so versucht das Programm nun durch Probieren aller möglichen Schlüssel den gesuchten zu finden. Dies kann je nach Rechner und Schlüssel jedoch sehr lange dauern. Aus diesem Grund kann dieser Ablauf per Escape-Taste oder durch Anwählen des Buttons ‚Abbrechen‘ beendet werden.

f) Das Menü Hilfe

In diesem Menü sind einige sehr rudimentäre Hilfsfunktionen zu finden:

Durch Anwählen des Menüpunktes ‚Inhalt‘ wird eine Windows-Hilfe aufgerufen, in welchem die Bedienung des Programmes beschrieben wird.

Wird der Menüpunkt ‚Über Datenterminal‘ gewählt, so erscheint eine Dialogbox, in welchem kurze Informationen über dieses Programm (Versionsnummer, Autor) angezeigt werden.

B – Die Bedienung des Leitungsverschlüsslers

Die Bedienung des eigentlichen Leitungsverschlüsslers gestaltet sich einfach. Die in ein Kunststoffgehäuse eingebaute Mikrocontrollerschaltung besitzt nur wenige für den Benutzer zur Verfügung stehende Bedienelemente.

a) Die Verbindungskabel

Wichtigstes Element ist die serielle Schnittstelle, die in einem 9-poligen Anschluss implementiert ist. Hierbei ist darauf zu achten, dass über diesen Anschluss sowohl die erste als auch die zweite serielle Schnittstelle des Mikrocontrollers nach außen geführt werden, so dass ein spezielles Anschlusskabel verwendet werden muss. Die Schnittstelle hat folgende Belegung:

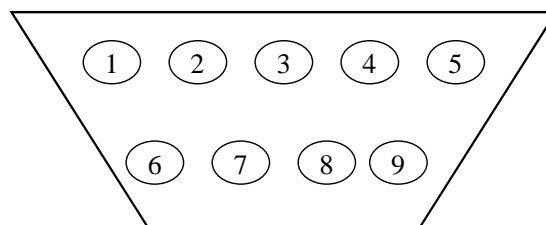


Abb. 13 - Der 9-polige SUB-D-Stecker an der Mikrocontrollerschaltung

Die einzelnen Anschlüsse haben folgende Belegung (aus [Ra1999]):

- | | |
|---|--------------------------------------|
| 1 | nicht angeschlossen |
| 2 | Empfangsdaten RxD, 1. Serieller Port |
| 3 | Sendedaten TxD, 1. Serieller Port |
| 4 | B-Leitung für RS485-Schnittstelle |
| 5 | Signal- / Betriebserde GND |
| 6 | nicht angeschlossen |
| 7 | Sendedaten TxD, 2. Serieller Port |
| 8 | Empfangsdaten RxD, 2. Serieller Port |
| 9 | A-Leitung für RS485-Schnittstelle |

Es ist also ein spezielles Verbindungskabel notwendig, welches die entsprechenden Leitungen auf einer Steckerbuchse auf die zwei Kabel verteilt. Für den Masseanschluss muss für beide Kabel der gleiche Anschluss genutzt werden. Des weiteren muss darauf geachtet werden, dass

die Leitungen wie in einem Nullmodemkabel geführt werden müssen, d. h. die Sendeleitung des einen Anschlusses muss mit der Empfangsleitung des anderen Anschlusses verbunden werden und umgekehrt. Da auf einem PC auch noch die Hardware-Handshake-Leitungen abgefragt werden, müssen diese Leitungen gebrückt werden, womit sich bei der Verwendung nur eines Leitungsverchlüssels folgende Leitungsführung ergibt:

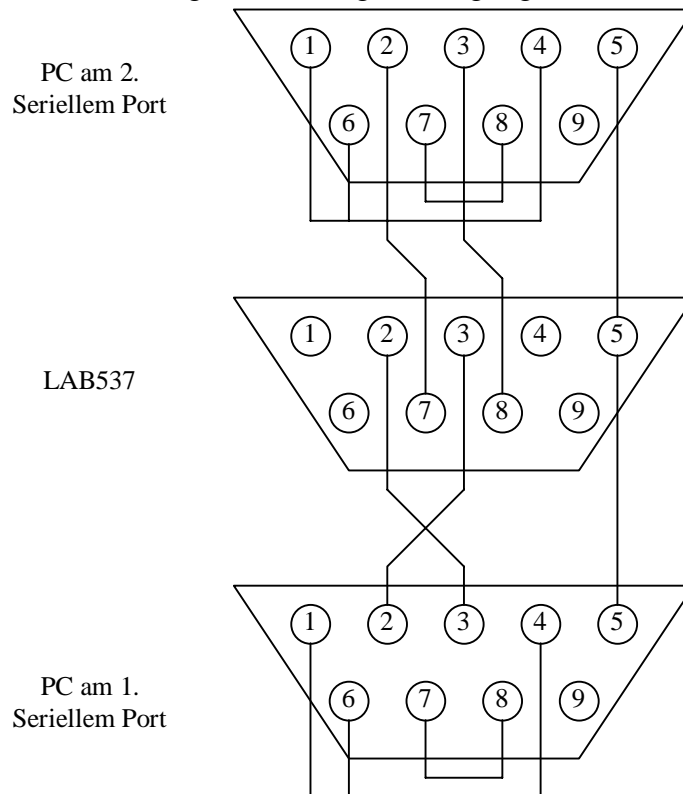


Abb. 14 - Das serielle Kabel (aus [Ra1999])

Die Anschlussbelegung für den Stecker an der Mikrocontrollerschaltung LAB537 ist bereits weiter oben in diesem Kapitel beschrieben, für die seriellen Schnittstellen eines PCs gilt folgende Pinbelegung:

- 1 Empfangspegel DCD (Data Carrier Detect)
- 2 Empfangsdaten RxD
- 3 Sendedaten TxD
- 4 Terminal betriebsbereit DTR (Data Terminal Ready)
- 5 Signal- / Betriebserde GND
- 6 Betriebsbereitschaft DSR (Data Set Ready)
- 7 Sendeteil ein RTS (Read To Send)
- 8 Sendebereitschaft CTS (Clear To Send)
- 9 Ankommender Ruf RI (Ring Indicator)

b) Stromversorgung

Die Schaltung besitzt auf ihrer Anschlussseite eine Steckerbuchse, an welche ein Netzteil mit einem 2,5mm Klinkenstecker angeschlossen werden muss, welche die Schaltung mit ca. 7,5 V Gleichspannung versorgt. Es sollte folgende Polung verwendet werden:

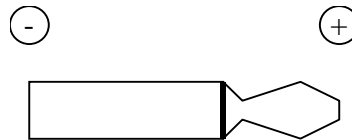


Abb. 15 - Die Stromversorgung

c) Betrieb des Leitungsverchlüsslers

Prinzipiell kann der Leitungsverchlüssler auf zwei verschiedene Arten betrieben werden:

Zum einen kann das mit der Schaltung ausgelieferte CESY-Betriebssystem und das CESY-Entwicklungssystem verwendet werden. Hierbei muss das Verschlüsselungsprogramm zunächst in den RAM-Speicher der Schaltung geladen werden, was mit Hilfe des genannten CESY-Entwicklungssystems und einem wie oben aufgebauten seriellen Übertragungskabel sehr leicht möglich ist. Es muss allerdings darauf geachtet werden, dass das zu übertragende Programm auch im Binärformat im Speicher des Entwicklungssystems vorliegt, d. h. es muss zunächst kompiliert bzw. assembliert werden. In einem Hex-Monitor kann das Programm dann auch noch nachbearbeitet werden.

Nach der Übertragung kann das Verschlüsselungsprogramm dann durch Betätigen der rechten Reset-Taste (Software-Reset) gestartet werden.

Eine weitere Möglichkeit besteht darin, das Mikrocontroller-Programm auf ein Eprom zu brennen und dieses an der Stelle des Eproms mit dem CESY-Betriebssystem in die Schaltung einzubauen. In diesem Fall wird das Verschlüsselungsprogramm sofort nach dem Einschalten der Stromzufuhr gestartet.

In beiden Fällen ist darauf zu achten, dass zunächst ein Schlüssel in den Leitungsverchlüssler übertragen werden muss, bevor mit der Übertragung der eigentlichen Daten begonnen werden kann.



Abb. 16 – Der betriebsbereite Leitungsverchlüssler

Abbildungsverzeichnis

<i>Abb. 1 - Das Schema der Leitungsverchlüsselung</i>	3
<i>Abb. 2 - Die Verschlüsselung</i>	9
<i>Abb. 3 - Ein Brute-Force-Angriff mit dem Programm 'Datenterminal'</i>	12
<i>Abb. 4 - Aufbau eines Datenframes</i>	16
<i>Abb. 5 - Die Endekennung</i>	16
<i>Abb. 6 - Versenden von Datenpaketen</i>	17
<i>Abb. 7 - Der Kommunikationsablauf</i>	18
<i>Abb. 8 - Blockschaltbild des MC80C537 (aus [MüWa1999])</i>	19
<i>Abb. 9 - Die Mikrocontrollerschaltung</i>	20
<i>Abb. 10 - Der Ablauf des Mikrocontroller-Programmes</i>	25
<i>Abb. 11 - Das Klassenmodell (aus [LeAr1998])</i>	36
<i>Abb. 12 - Das Datenterminal-Programm mit Dialog zum Versenden</i>	41
<i>Abb. 13 - Der 9-polige SUB-D-Stecker an der Mikrocontrollerschaltung</i>	42
<i>Abb. 14 - Das serielle Kabel (aus [Ra1999])</i>	43
<i>Abb. 15 - Die Stromversorgung</i>	44
<i>Abb. 16 - Der betriebsbereite Leitungsverchlüsseler</i>	44

Glossar

Acknowledge – ACK, NAK

Verschiedene Übertragungsprotokolle sind darauf aufgebaut, dass der Empfänger zur Bestätigung des Empfangs einer Nachricht dem Sender ein Acknowledge (ACK) zurücksendet. Meistens wird an die Nachricht auch noch eine Prüfsumme angehängt, die nach dem Empfang erneut berechnet wird um sie vergleichen zu können. Wird dabei ein Unterschied zwischen empfangener und berechneter Prüfsumme festgestellt, so wird dem Sender ein negatives Acknowledge (NAK) zurückgesendet.

Autoreload-Modus

Die Timer eines Mikrocontrollers der 8051-Familie lassen sich im sogenannten Autoreload-Modus betreiben. Hierbei wird der im High-Byte des Timers gespeicherte Wert ins Low-Byte kopiert, wenn dieses den Wert ‚0‘ erreicht hat. Der Timer kann dabei allerdings nur als 8-Bit-Timer genutzt werden. Dieser Modus wird verwendet um aus dem Prozessortakt andere Takt-raten zu gewinnen, z. B. für die Baudrate einer seriellen Übertragung.

Blockchiffre

Bei einer Blockchiffre werden mehrere Datenbytes zu einem Block zusammengefasst und als eine Einheit verschlüsselt. Dies hat den Vorteil, dass bei einem Textdokument mehrere Buchstaben zusammen verschlüsselt werden und somit die Häufigkeitsverteilung der Buchstaben, die charakteristisch für bestimmte Sprachen ist, verschleiert werden. Außerdem ergibt sich ein Geschwindigkeitsvorteil, wenn auf 32- oder 64-Bit-Rechner direkt Blöcke von entsprechend 32 oder 64 Bit verschlüsselt werden.

Brute-Force-Angriff

Der Brute-Force-Angriff ist eine Methode um den Schlüssel eines verschlüsselten Textes zu finden, indem der Reihe nach alle möglichen Schlüssel ausprobiert werden. Hierbei ist neben dem Verschlüsselungsalgorithmus meist auch noch ein kleiner Teil des entschlüsselten Textes bekannt, so dass nach einem Entschlüsselungsvorgang verglichen werden kann, ob es sich um den richtigen Schlüssel handelt.

EOT – end of text

Am Ende der kompletten Übertragung wird in manchen Protokollen ein sogenanntes EOT-Byte gesendet, um das Ende kenntlich zu machen. Hieran kann der Empfänger erkennen, dass die Übertragung nun beendet ist und keine weiteren Daten mehr zu erwarten sind.

Message Map

Bei der Verwendung der MFC zur Programmierung mit Visual C++ ist es für den Programmierer nicht erforderlich, eine eigene Nachrichtenschleife zu implementieren, in welcher er auf alle Windows-Nachrichten (Ereignisse / Events) reagiert. Er muss stattdessen eine sogenannte Message Map anlegen, in welcher allen Ereignissen, die meist durch eine ID gekennzeichnet sind, eine Methode zugeordnet werden kann, die beim Eintreten des Ereignisses abgearbeitet werden soll. Die eigentliche Abfrage der Ereignisse findet dann innerhalb der MFC statt.

MFC – Microsoft Foundation Classes

Die Microsoft Foundation Classes bestehen aus einem Klassenpaket für MS-Visual C++, die dem Programmierer eine etwas einfachere Schnittstelle zu den Windows-APIs zur Verfügung stellen. Der Programmierer nutzt also nicht mehr direkt diese APIs, sondern die entsprechenden Funktionen bzw. Klassen der MFC, welche die Aufrufe in die gewünschten API-Calls umsetzen.

Mikrocontroller

Ein Mikrocontroller ist ein Mikrocomputer, der komplett in einem einzigen IC untergebracht ist. Neben einem Prozessor sind dort z. B. auch noch Datenspeicher (RAM) oder verschiedene Schnittstellen (parallele oder serielle, A/D-Wandler u. a.) untergebracht. Manche Mikrocontroller-Versionen besitzen sogar einen eigenen internen Programmspeicher, der entweder als wirkliches ROM oder aber auch als Eprom enthalten sein kann. Zu den bekanntesten Mikrocontrollern gehören die der i8051-Familie, die in verschiedenen Ausführungen und von verschiedenen Herstellern erhältlich sind.

Port

Die Mikrocontroller der i8051-Familie enthalten als Schnittstellen die sogenannten Ports, auf welchen Daten parallel eingelesen oder ausgegeben werden können. Einige dieser Ports werden bei den meisten Schaltungen zur Kommunikation mit externem Programm- bzw. Datenspeicher verwendet, andere können als serielle Schnittstelle oder zur parallelen Datenausgabe genutzt werden.

Protokoll

In der Datenübertragung versteht man unter einem Protokoll die Art und Weise, in welche die Daten übertragen werden. Im Protokoll wird vorgeschrieben, welche zusätzlichen Informationen in einem Datenpaket enthalten sein müssen, welche Größe die einzelnen Datenpakete maximal haben können oder wie der Empfänger auf den Empfang eines Datenpaketes reagieren muss.

Prüfsumme

Bei den meisten Protokollen muss zusätzlich zu den Daten auch noch eine Prüfsumme mitgesendet werden, damit der Empfänger sofort prüfen kann, ob die Daten korrekt angekommen sind. Diese Prüfsumme hat eine feste Länge, der Wert wird aus den zugehörigen Daten berechnet. Ein bekanntes Verfahren zur Berechnung einer Prüfsumme ist der sogenannte CRC (Cyclic Redundancy Check).

RC5-Algorithmus

Der RC5-Algorithmus ist ein symmetrischer Blockchiffre, der von Ron Rivest der RSA-Laboratories entwickelt wurde. Schlüssel- und Blocklänge sowie Rundenzahl, d. h. Anzahl der durchzuführenden Verschlüsselungen können per Parameter festgelegt werden. So verwendet der RC5-64 eine Schlüssellänge von 64 Bit mit 12 Verschlüsselungsrunden. Ein weiterer Vorteil dieses Algorithmus ist die Verwendung der einfachen Operationen Exklusives Oder, Addition und Bitrotationen, so dass die Verschlüsselung sehr schnell abläuft.

Schlüsselraum

Unter dem Schlüsselraum versteht man die Menge aller möglichen Schlüssel eines Verschlüsselungsalgorithmus. Dieser wird durch die Schlüssellänge angegeben. So benutzt ein Verschlüsselungsalgorithmus, der eine Schlüssellänge von 64 Bit verwendet, einen Schlüsselraum, der Schlüssel von 0 bis 2^{64} (= 18.446.744.073.709.551.616, ca. 18,5 Milliarden) umfaßt.

Stromchiffre

Ein Stromchiffre ist ein Verschlüsselungsalgorithmus, der jedes Zeichen eines Zeichenstromes unabhängig von anderen Zeichen verschlüsselt. Er stellt das Gegenstück zu einem Blockchiffre dar, bei dem immer nur ein kompletter Zeichenblock verschlüsselt werden kann, wobei die Nachricht meist auf ein ganzzahliges Vielfaches der Blocklänge aufgefüllt werden muss. Ein Block besteht meist aus vier oder acht Zeichen.

Vaterklasse

In einer objektorientierten Programmiersprache versteht man unter der Bezeichnung Vaterklasse diejenige Klasse, von der eine neue Klasse abgeleitet wurde. Die neue Klasse erbt dann alle Eigenschaften (Attribute, Methoden) ihrer Vaterklasse. Aus diesem Grund kann in einem Objekt der neuen Klasse auch auf die Methoden der Vaterklasse zugegriffen werden.

Index

A

Acknowledge · 14, 19, 28, 44
Akku · 21
Anschlusskabel · 40
Assembler · 19
Autoreload-Modus · 20, 44

B

Baudrate · 20, 25, 28, 31
Blockchiffre · 44
Brute-Force-Angriff · 10, 39, 44

C

C++ · 24
Carry-Flag · 14
Client-Bereich · 26
COM-Port · 28, 29, 31, 38
C-String · 35

D

Dateiauswahldialog · 37
Dateizeiger · 33
DES · 36
Dialogfeld · 32
Drucker · 26, 37
Druckereinrichtung · Siehe Drucker
Druckvorschau · 27, 37

E

Editierfeld · 31
Empfangen · 28, 38
Entschlüsselungsalgorithmus · 22
EOT · 14, 44
Eprom · 18, 42
Eprom-Simulator · 18

F

Font · 27, 30

H

Halbduplex · 15
Handshake · 41
Häufigkeitsverteilung · 8
hexadezimal · 25

I

Internet · 6

M

Member-Variablen · 32
Message Map · 44, 32
MFC · 24, 45
Mikrocontroller · 17, 19, 36, 45

N

NAK · 15, 28, 29
Nullmodemkabel · 40

O

OnTimer · 29

P

Port · 19, 45
Protokoll · 13, 24, 29, 45
Prüfbyte · *Siehe Prüfsumme*
Prüfsumme · 14, 19, 22, 28, 45

R

RAM · 42
RC5-Algorithmus · 10, 39, 45
Rotation · 8
RS232-C · 6
RxD · 40

S

SBUF · 21
Schlüsselbyte · 7, 8, 25, 28
Schlüsselraum · 10, 45
SCON · 21
seriell · 19, 26, 28, 31, 40
Stromchiffre · 8, 46
symmetrische Verschlüsselung · 7

T

Timer · 26, 29
TxD · 40

Ü

Übertragungsgeschwindigkeit · 29, 38
Übertragungsqualität · 28, 29

V

V.24 · 6
Vaterklasse · 26, 27, 46
Verschiebe-Chiffre · 9
Verschlüsselungsalgorithmus · 7, 10, 22
Versenden · 28, 38

Vigenère-Chiffre · 9

X

X-Oder-Operation · 7

Z

Zeichenkette · 9
Zeitüberschreitung · 29

Quellenverzeichnis

[MüWa1999]

Müller, Helmut; Walz, Lothar
Mikroprozessortechnik
Würzburg [Vogel] ⁵1999

[LeAr1998]

Leinecker, Richard C.; Archer, Tom
Die Visual C++ Bibel
Übersetzung aus dem Amerikanischen von G&U Technische Dokumentation GmbH
Bonn [MITP] ¹1998

[Mi1998]

Microsoft
msdn Library – Visual Studio 6.0
Mitte 1998

[Ra1999]

Rakers, Sven
LAB 537 – Aufbauanleitung und Referenzhandbuch
Münster 1999
<http://www.rakers.de>