

Projektdokumentation des C-Programms

„CryptIt“



von Tom Schreiber (Seminargruppe IF04w1)

März 2006, Mittweida

Zusammenfassung

Bei dem vorliegenden Programm handelt es sich um ein Kommandozeilentool, mit welchem es möglich ist einzelne Dateien mittels eines Passwortes zu verschlüsseln und zu entschlüsseln. Dieses Verschlüsselungsprogramm ist in der Sprache C geschrieben und für den Einsatz auf Linux - Systemen gedacht. Zur Verschlüsselung der Datei wird der „Blowfish“ Algorithmus von Bruce Schneier¹ verwendet. Hierbei handelt es sich um ein asynchrones Verschlüsselungsverfahren mit einer variablen Verschlüsselungstiefe von 32 - 448bitabhängig von der Länge des verwendeten Passwortes.

Die hier vorliegende Dokumentation gibt eine Funktionsübersicht des Programms und geht dabei auf Details der Implementierung und auf die Aufgaben der einzelnen Programmmodule ein. Diese Dokumentation richtet sich in erster Linie an Entwickler, welche den Quelltext verstehen möchten, um ihn für eigene Zwecke einzusetzen oder ihn zu modifizieren. Der Quelltext des vorliegenden Programms kann im Internet auf der Homepage von Tom Schreiber² herunter geladen werden und frei verwendet werden.

Eine kurze Bedienungsanleitung des Programms finden sie in der Datei „Readme.txt“, welche dem Programm beiliegt.

¹ weitere Informationen finden sie auf der Homepage von Bruce Schneier unter <http://www.schneier.com/blowfish.html>

² die Projektseite von CryptIt finden sie unter <http://cryptit.tomhost.de>



Inhaltsverzeichnis

1. Grundlegendes	4
1.1 Programmfunktion.....	4
1.2 Informationen zur Kompilierung	4
1.2 Programmstruktur	5
2. Die Ver-/Entschlüsselungsmodule	6
2.1 Der Blowfish – Algorithmus	6
2.2 Das Blowfish – Verschlüsselungsmodul (blowfish.c)	7
2.3 Das Kryptografiemodul (crypt.c)	7
2.3.1 Der Dateiheader	8
2.3.2 Der Passwort-Prüfblock	8
2.3.3 Das Kryptografiehilfsmodul (crypt-tools.c)	9
3. Weitere Programmmodule	9
3.1 Das Hauptmodul (main.c)	9
3.2 Das Hilfsmodul (helpers.c)	9
3.3 Das Fehlerausgabemodul (error.c)	10
4. Programmausblick	10
4.1 Fehlerquellen	10
4.2 Mögliche Erweiterungen und Verbesserungen	10
4.3 Zusammenfassung	11
Anhang	11
A Selbständigkeitserklärung	12
B Quellen und Hilfsmittel	12
B.1 Literatur	12
B.2 Internet	12

1. Grundlegendes

1.1 Programmfunktion

Bei dem vorliegenden Programm handelt es sich um ein Kommandozeilentool. Dieses Tool gestattet es eine einzelne Datei mittels eines Passwortes (4 bis max. 56 Zeichen) zu verschlüsseln und wieder zu entschlüsseln. Als Verschlüsselungsalgorithmus wird der symmetrische „Blowfish“ Algorithmus verwendet. Die Funktion des Programms wird über Parameter gesteuert.

Programmparameter

Aufrufmöglichkeiten

```
cryptit -c[n] [-pass:xxx] Datei1 [Datei2]
cryptit -d [-pass:xxx] Datei1 [Datei2]
cryptit -i Datei1
cryptit -v oder --version
cryptit -h oder --help
```

Bedeutung der Parameter

-c	Verschlüsseln
-d	Entschlüsseln
-cn	Verschlüsseln ohne Passwortprüfblock
-pass:xxx	Passwort xxx verwenden
Datei1	Eingangsdatei
Datei2	Ausgangsdatei; wird keine Ausgangsdatei angegeben so wird die Eingangsdatei überschrieben
-i	Informationen/Headeranalyse einer verschlüsselten Datei
-v; --version	Versionsinformationen anzeigen
-h; --help	Hilfe anzeigen

1.2 Informationen zur Kompilierung

Das Programm läuft auf einer standardmäßigen Linux/Unix-Umgebung. Es werden nur C – Standardbibliotheken verwendet. Als Compiler wurde der gcc genutzt, andere Compiler dürften aber keine Probleme bereiten, da sich an die ANSI – C - Standardisierung gehalten wurde. Der Quelltext kann für eigene Programme verwendet und modifiziert werden. Der Quellcode liegt im weltweiten Netz auf der Homepage des Autors zum Download bereit [i0].

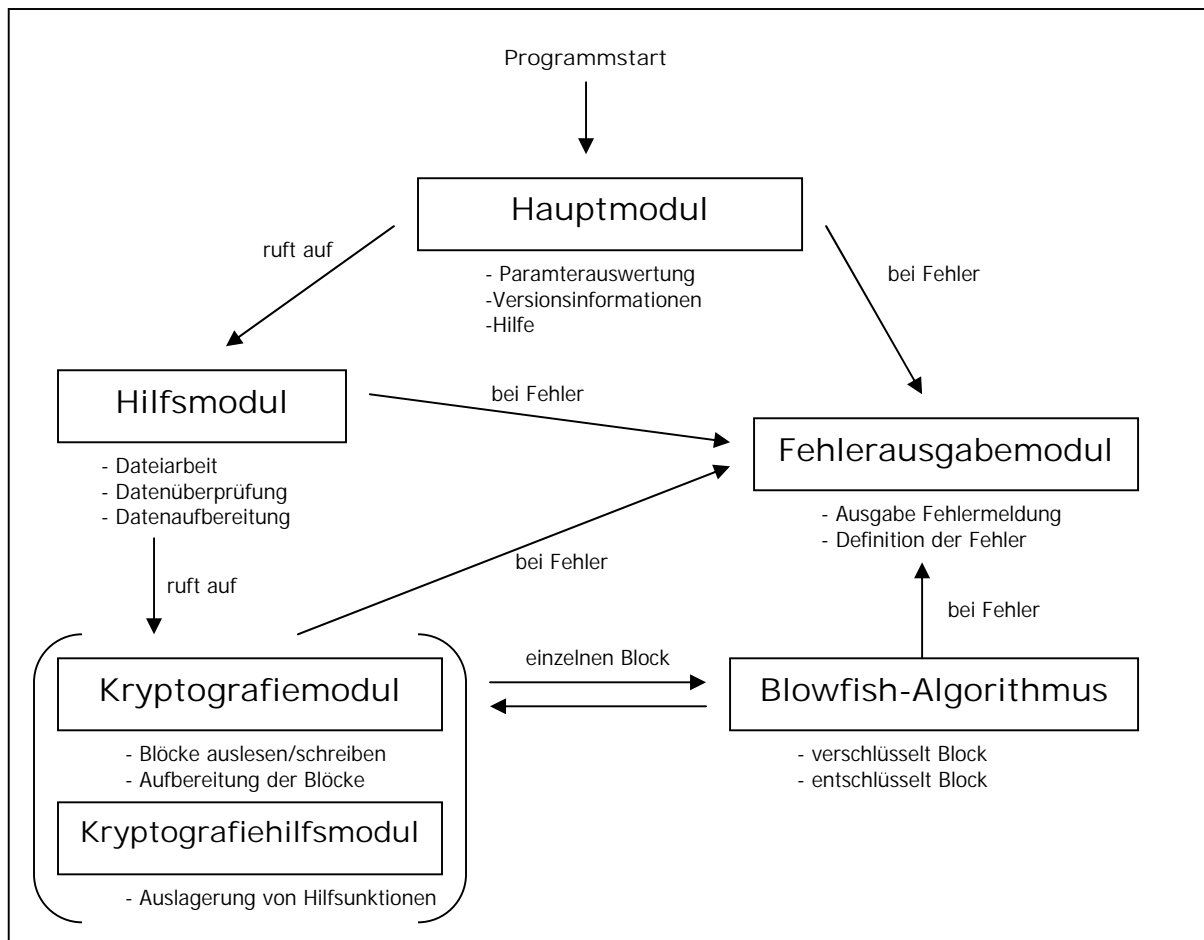
Zur Kompilierung des Programms kann das make Tool verwendet werden, das Makefile ist folgendermassen vorkonfiguriert:

make Optionen

make	kompiert CryptIt mit Debuginformationen als „cryptitd“
make release	kompiert CryptIt ohne Debuginformationen als „cryptit“
make package	erstellt ein Archiv mit den aktuellen Quellcodedateien und der binären „cryptit“ im Verzeichnis /dist
make binpackage	erstellt ein Archiv der binären „cryptit“ und der README.TXT im Verzeichnis /dist/bin
make clean	bereinigt das Verzeichnis

1.2 Programmstruktur

Um das Programm übersichtlich und leicht wartbar zu gestalten wurde von Anfang auf eine strikte Trennung der einzelnen Komponenten geachtet. Das Programm besteht aus 6 Modulen, wobei jedes dieser Module fest definierte Aufgaben übernimmt. Der grobe Zusammenhang zwischen den einzelnen Modulen ist in der folgenden Grafik dargestellt.



Die folgende Übersicht gibt an, welche Quellcode - Dateien den einzelnen Modulen zuzuordnen sind:

Modulname	Zugehörige Dateien
Hauptmodul	main.c, main.h
Hilfsmodul	helpers.c, helpers.h
Fehlerausgabemodul	error.c, error.h
Kryptografiemodul	crypt.c, crypt.h
Kryptografiehilfsmodul	crypt-tool.c, crypt-tools.h
Blowfish-Verschlüsselungsmodul	blowfish.c, blowfish.h, blow_const.h

Bei dem Programm handelt es sich um eine ausführbare Kommandozeilenanwendung, welche über Parameter gesteuert wird. Die Parameterevaluation und Parameterüberprüfung findet im Programmeinstiegspunkt, der main - Methode, statt. Die main - Methode ruft, je nach Parameter, die erforderlichen Funktion mit den

zugehörigen Parametern auf. Der Aufruf des Kryptografiemoduls erfolgt über das Hilfsmodul, welches die Daten aus den Parametern überprüft, aufbereitet und gegebenenfalls ergänzt. Für eventuell auftretende Fehler gibt es ein Fehlermodul, welches eine Fehlerbeschreibung ausgibt und das Programm beendet. Das Fehlermodul wird von allen Modulen bzw. deren Funktionen in gleicher Weise genutzt.

Das eigentliche Herzstück des Programms ist das Kryptografiemodul zusammen mit dem Blowfish - Algorithmusmodul. Diese Module sind für die Ver- und Entschlüsselung der Datei zuständig. Im nächsten Abschnitt wird deshalb detailliert auf diese Module eingegangen. Anschließend wird auf die anderen Programmmodule im darauffolgenden Abschnitt kurz eingegangen.

2. Die Ver-/Entschlüsselungsmodule

2.1 Der Blowfish – Algorithmus

Bei dem verwendeten Blowfish - Algorithmus handelt es sich um einen symmetrischen Verschlüsselungsalgorithmus, welcher 1993 von Bruce Schneier entwickelt wurde. Die Verschlüsselungstiefe beträgt 32 - 448bit je nach der Länge des verwendeten Passwortes, welches dementsprechend mindestens 4 Zeichen, höchsten jedoch 56 Zeichen lang sein muss. Der Algorithmus ist nicht patentiert und kann frei verwendet werden, dies ist auch ein Grund warum die Wahl auf gerade diesen Algorithmus fiel. Ein weiterer Grund für die Benutzung dieses Algorithmus war das er auf 32bit Systemen ziemlich schnell arbeitet und relativ einfach in C zu implementieren ist. Auf die Details des Algorithmus soll hier nicht weiter eingegangen werden, der Interessierte findet auf verschiedenen Seiten im weltweiten Netz umfangreiche weitergehenden Informationen, als Einstieg sei hier [i1] und [i2] genannt. Im weiteren nur ein kurzer Abriss des Algorithmus.

Die Grundlage bilden so genannte „Boxes“ (Tabellen), eine P-Box mit 18 Einträgen und vier S-Boxes jeweils 256 Einträgen, jeder dieser Einträge ist 32bit groß. Diese werden als Array implementiert und mit der hexadezimalen Darstellung der Zahl Pi gefüllt. Im ersten Schritt werden diese Boxes mit dem Passwort, dem Key, initialisiert (blowfish_init). Wurden die Boxes initialisiert kann mit der eigentlichen Ver- und Entschlüsselung der Informationen begonnen werden. Der Algorithmus verarbeitet 64bit Blöcke. Dieser Block wird in zwei 32bit Blöcke aufgeteilt (Block1 und Block2) und dem Algorithmus übergeben. Die Blöcke werden nun mit Hilfe der Boxes verschlüsselt

```
for( i von 0 bis 15)
    Block1 = Block1 xor Pbox[i]
    Block2 = Block2 xor f(Block1)
    vertausche Block1 und Block2
next i

vertausche Block1 und Block2
Block2 = Block2 xor PBox[16]
Block1 = Block1 xor PBox[17]
```

Verschlüsselungsalgorithmus (Pseudocode)

Die Funktion f bindet die Sboxes mit in den Algorithmus ein. In dieser Funktion wird zunächst der übergebene Block in 4 Teile zu je 8bit zerlegt (hier a, b, c, d), der Rückgabeblock brechnet sich wie folgt.

$$\text{RückgabeBlock} = f(\text{Block}) = ((\text{SBox1}[a] + \text{SBox2}[b]) \text{ xor } \text{SBox3}[c]) + \text{SBox4}[d]$$

Die Entschlüsselung erfolgt fast analog zur Verschlüsselung, wobei die Boxes zuerst mit dem Passwort initialisiert werden müssen und dann die blockweise Entschlüsselung begonnen werden kann. Auf eine Beschreibung wird hier verzichtet, die Umsetzung kann im Quelltext eingesehen werden oder auf den oben genannten Internetseiten nachgelesen werden.

2.2 Das Blowfish – Verschlüsselungsmodul (blowfish.c)

Der Implementierung dieses Algorithmus in der Sprache C entspricht weitestgehend der Musterimplementierung von Paul Kocher [i4]. Die 5 Boxes mit der Zahl Pi wurden von Bruce Schneider übernommen [i5]. Eine Besonderheit die man in C beachten muss, ist die Handhabung der Boxes. Dazu muss die aufrufende Funktion eine Kopie dieser Boxes verwenden und diese beim Initialisieren und Ver-/entschlüsseln mit angeben (als Referenz). Bei objektorientierten Sprachen kann auf diesen Umstand verzichtet werden.

Wie bereits erwähnt arbeitet der Algorithmus mit einen 64bit Block, welcher in zwei 32bit Blöcke aufgeteilt wird. Als Datentyp für einen solchen Teilblock bietet sich demnach unsigned long an, welcher 4 Byte groß ist. Die Operationen und der Implementierungsumfang werden so vereinfacht, da auf die Definition eines eigenen Datentyps verzichtet werden kann.

Dieses Modul bildet das Kernstück des Programms. Der Algorithmus an sich ist allerdings nur in der Lage zwei long – Werte zu ver- bzw. zu entschlüsseln. Um die Arbeit mit Dateien zu ermöglichen muss also ein Modul vorangestellt werden, welches die Aufbereitung und Überprüfung der Daten übernimmt welche an dieses Verschlüsselungsmodul übergeben werden.

2.3 Das Kryptografiemodul (crypt.c)

Dieses Modul ist eine Art Hülle welches um das Blowfish – Verschlüsselungsmodul gelegt wird, es sorgt für die Aufbereitung der Daten aus einer Datei und die korrekte Übergabe der Daten an den Algorithmus, es fängt Fehler ab schreibt die geänderten Daten zurück in eine andere Datei. Der Quelltext ist an den meisten Stellen selbsterklärend, auf einige Besonderheiten wird hier noch eingegangen.

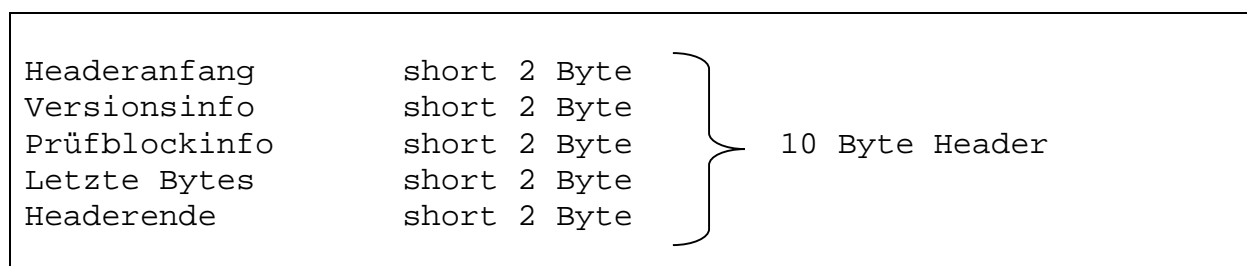
Das Blowfish – Verschlüsselungsmodul arbeitet ausschließlich mit unsigned long Werten. Beim Lesen aus einer Datei werden immer 4Byte ausgelesen und als ein unsigned long - Wert behandelt. Zwei aufeinander folgende ausgelesene unsigned long Werte bilden einen Datenblock und werden an das Blowfish – Verschlüsselungsmodul übergeben.

Da der Blowfish – Algorithmus mit 8Byte Blöcken arbeitet, ergibt sich ein Problem bezüglich einer zu verschlüsselnden Datei. Diese ist nämlich in den meisten Fällen nicht so einfach in 8Byte aufzuteilen, es bleibt meistens ein Rest an Bytes übrig. Die Anzahl dieser letzten Bytes werden im Programm die lastBytes genannt. Diese Anzahl ist über eine einfache Modulfunktion mit Hilfe der Dateigröße leicht zu berechnen. Diese letzten Bytes werden in einen Nullblock (mit 00H initialisierter Block) eingeschoben und bilden somit den letzten 8 Byte Block, welcher nun regulär verschlüsselt werden kann. Bei der Entschlüsselung muss die Anzahl dieser letzten Bytes zum Zeitpunkt der Verschlüsselung bekannt sein, damit die Daten ordnungsgemäß wiederhergestellt werden können. Eine Ermittlung über die Dateigröße funktioniert jetzt nicht mehr, da die verschlüsselte Datei immer aus ganzen 8Byte Blöcken besteht. Diese Information muss sozusagen mit der verschlüsselten Datei zusätzlich übergeben werden, somit ist die Einführung eines Dateikopfes (Header), welcher diese Informationen enthält, unvermeidbar.

2.3.1 Der Dateiheder

Die verschlüsselte Datei bekommt einen Dateiheder, welcher zusätzliche Informationen für das Entschlüsseln bereitstellt. Diese Informationen umfassen die Version von CryptIt mit der die Datei verschlüsselt wurde, ob ein Passwortprüfblock vorhanden ist (dazu im nächsten Abschnitt mehr) und wie viele Bytes im letzten Block zur Originaldatei gehören. Die einzige Information, welche zum Enkodieren zwingend notwendig ist, ist die Anzahl der letzten Bytes der Original Datei. Damit der Header als gültig erkannt wird, steht am Anfang und am Ende des Headers noch ein willkürlich gewählter short Wert (definiert in crypt.h), welcher einen Header – Rahmen bildet.

Mittels der Versionsinformation kann überprüft werden, ob die Datei eventuell mit einer neueren Programmversion verschlüsselt wurde, die unter Umständen einen veränderten Algorithmus verwendet und somit eine korrekte Entschlüsselung nicht mehr gewährleistet ist. Andersherum kann in einer neueren Programmversion mittels dieser Versionsinformation die Abwärtskompatibilität realisiert werden.



Header (die ersten 10 Byte einer verschlüsselten Datei)

Mit dem Parameter `-i` ist es möglich den Header einer verschlüsselten Datei zu überprüfen und auszuwerten. Dies geschieht mittels der Funktion `analyse_header` im Kryptografiehilfsmodul.

2.3.2 Der Passwort-Prüfblock

Damit bei der Entschlüsselung festgestellt werden kann, ob das eingegebene Passwort überhaupt mit dem bei der Verschlüsselung verwendeten Passwort übereinstimmt wird standardmäßig ein Prüfblock bei der Verschlüsselung erstellt.

Dabei handelt es sich um ein willkürlich festgelegten Referenzblock, welcher fest im Programm definiert ist (in der Datei crypt.h). Dieser Block wird beim Verschlüsseln mit dem Algorithmus regulär verschlüsselt und als erster Block hinter den Header geschrieben. Beim Entschlüsseln wird anhand dieses Blockes die Korrektheit des Passwortes überprüft, da dem Programm bekannt ist welche Daten der Block nach seiner Entschlüsselung mit dem korrekten Passwort beinhalten müsste.

Die Erstellung dieses Blockes kann mit dem zusätzlichen Parameter n unterdrückt werden. Ohne diesen Prüfblock wird ein Brute - Force Angriff auf die Datei erheblich erschwert, da dem Angreifer nun der Inhalt der Datei (oder zumindest ein Teil davon) bekannt sein müsste. Das Programm entschlüsselt die Datei streng nach dem Algorithmus, da eine Überprüfung des Passwortes ohne den Prüfblock nicht möglich ist. Stimmt das Passwort nicht mit dem bei der Verschlüsselung angegebenen überein kommt schlicht Datenmüll heraus. Ob ein solcher Prüfblock vorliegt oder nicht, steht im Header der verschlüsselten Datei (siehe Abschnitt 2.3.1).

2.3.3 Das Kryptografiehilfsmodul (crypt-tools.c)

In das Kryptografiehilfsmodul wurden einige Funktionen aus dem Kryptografiehauptmodul ausgelagert, in erster Linie um die Übersichtlichkeit zu gewährleisten. Die Funktionen umfassen die Anzeige der Headerinformationen (Parameter -i), die Überprüfung der Passwortlänge und die Rückgabe der Dateigröße. Die Kommentare im Quelltext sollte an dieser Stelle zum Verständnis genügen.

3. Weitere Programmmodule

3.1 Das Hauptmodul (main.c)

Im Hauptmodul (main.c) steht als Programmeinstiegspunkt die main - Methode. In dieser werden die übergebenen Parameter ausgewertet und die entsprechenden Funktionen aufgerufen. Hier erfolgt noch keine Überprüfung auf die Gültigkeit der Parameter, zum Beispiel das Vorhandensein der Eingangsdatei oder die Länge des Passwortes. Die Überprüfung auf die Gültigkeit erfolgt erst in den entsprechenden Funktionen, welche eventuell auftretende Fehler entsprechend behandeln bzw. dem Benutzer melden. Für die Datenüberprüfung stellt das Hilfsmodul Funktionen bereit. Das Hauptmodul verfügt ebenfalls noch über eine Funktion zum Anzeigen der Versionsinformationen und zur Anzeigen der Hilfe.

3.2 Das Hilfsmodul (helpers.c)

Das Hilfsmodul stellt zwei wichtige Funktionen bereit, welche die übergebenen Parameter auf Richtigkeit überprüfen. Weiterhin ergänzen diese, eventuell noch fehlende notwendige Informationen für die Ver- bzw. Entschlüsselung. Die Funktionen prepare_crypt und prepare_decrypt erledigen außerdem die Dateiarbeit und übergeben dem Kryptografiemodul alle notwendigen Informationen für die Ver-/Entschlüsselung.

Falls das Passwort nicht über den Parameter angegeben wurde, erfragt die Funktion dieses. Wurde nur eine Datei angegeben, so wünscht der Nutzer, dass die

Eingangsdatei mit der verschlüsselten bzw. entschlüsselten Ausgabe überschrieben wird. Da die Ver-/Entschlüsselungsfunktion des Kryptografiemoduls aber zwei Dateien (Eingabe- und Ausgabedatei) zwingend voraussetzt, wird eine temporäre Datei angelegt (namens `~crypt_out.tmp`), welche aus Ausgabedatei dient. Im Anschluss an die Ver-/Entschlüsselung wird mittels der Funktion `replace_file` die Eingangsdatei durch diese temporäre Datei ersetzt.

3.3 Das Fehlerausgabemodul (`error.c`)

Dieses Modul besitzt nur einzige Funktion namens `err_print`, welche die Aufgabe hat eine detaillierte Fehlermeldung auf die Standard-Fehlerausgabe auszugeben und das Programm zu beenden. Die Definitionen der Fehlermeldung findet man in der Headerdatei `error.h`, in dieser stehen alle möglichen Fehlermeldungen des gesamten Programms. Die anderen Module rufen bei einem auftretenden Fehler lediglich die Funktion `err_print` mit der dazugehörigen Definition auf. Dies hat den Vorteil, dass die Fehlermeldungen übersichtlich in einer Datei zu finden sind und in dem Module selbst keine Fehlerausgabe implementiert werden muss. Diese Zentralisierung der Fehlermeldungen erlaubt außerdem eine schnelle Anpassung an andere Sprachen.

4. Programmausblick

4.1 Fehlerquellen

Bei dem Entwurf und der Implementierung des Programms wurde sehr auf die Stabilität des Programms geachtet und auf eine umfassende Fehlererkennung geachtet, dennoch wird es noch nicht berücksichtigte Fehlerquellen geben. Die hauptsächliche Fehlerquelle liegt in der Arbeit mit Dateien. Bei der Arbeit mit Dateien gibt es eine Vielzahl von Szenarien, bei welchen es zu Problemen kommen kann: ungenügende Zugriffsrechte, die angegebene Datei wird bereits verwendet, es handelt sich nicht um eine reguläre Datei, während Schreibvorgang wird Programm terminiert... Viele dieser Szenarien wurden bereits im Programm berücksichtigt, jedoch nicht alle. Das Programm reagiert zum Beispiel nicht auf das Terminationssignal, wird das Programm mitten in einer Verschlüsselung terminiert, ist die Ausgabedatei unvollständig besitzt aber einen korrekten Header.

Die Fehlerbehandlung im Programm selbst weißt außerdem noch Lücken auf. Gibt es zum Beispiel Probleme beim Datenzugriff während einer Verschlüsselung, so erkennt das Programm dies zwar und beendet sich mit einer Fehlermeldung. Es entsteht aber ebenfalls eine unvollständige Ausgabedatei und die beiden verwendeten Dateien werden auch nicht über `fclose()` geschlossen. Hier müsste das Fehlermodul noch um zusätzliche Funktionen erweitert werden, die bei solchen Fällen korrekt reagieren, d.h. die unvollständige Ausgabedatei löschen und die offenen Dateien versuchen korrekt zu schließen.

4.2 Mögliche Erweiterungen und Verbesserungen

Eine denkbare Erweiterung bzw. Verbesserung für das Programm, wäre in wie im vorigen Abschnitt bereits erwähnt, eine Überarbeitung des Fehlerbehandlungsmoduls in Hinblick auf spezielle Funktionen für die korrekte Programmbeendigung bei eventuell auftretenden Dateifehlern. Eine engere Zusammenarbeit mit der Standard-`ein- und ausgabe` wäre ebenfalls eine nützliche Verbesserung. Dadurch wäre

das Programm in Verbindung mit Pipes einsetzbar und würde gut in Shellskripten einsetzbar sein. Aufgrund der Tatsache, dass nur C – Standardbibliotheken verwendet wurden, ist eine Portierung auf Windowssysteme ebenfalls denkbar und könnte dort ebenfalls als Kommandozeilentool von Nutzen sein.

4.3 Schlusswort

Dieses Projekt schrieb der Autor als Belegarbeit für das Fach „Maschinennahes Programmieren“ im Rahmen seines Informatik Studiums. Neben den praktischen Nutzen dieses Programms, erlaubt der Quelltext einen Einblick in die Materie der softwaremäßig umsetzen Kryptografie. Dieses Gebiet ist sehr umfangreich und bietet wesentlich mehr als der hier eingesetzte, vergleichsweise simple Algorithmus. Dieses Projekt kann als Einstieg in dieses sehr komplexe Gebiet dienen. Durch die Modularisierung des Programms ist es zum Beispiel denkbar andere Algorithmen einzubinden um so mit verschiedenen Verschlüsselungsvarianten zu Arbeiten. Der Quelltext ist frei verfügbar und es steht somit jeden offen dieses Programm für seinen eigenen Zwecke zu nutzen.

Anhang

A Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Mittweida, 16. März 2006

Tom Schreiber

B Quellen und Hilfsmittel

B.1 Literatur

- [1] Erik de Casto Lopo / Peter Aitken / Bradley L. Jones
"C – Programmierung für Linux" (Markt + Technik Verlag)
- [2] Martin Gräfe
„C und Linux“ (Hanser Verlag)
- [3] Henning Mittelbach
„Einführung in C“ (Fachbuchverlag Leipzig)

B.2 Internet

- [i0] Homepage von Tom Schreiber – Projekt 'CryptIt'
<http://cryptit.tomhost.de>
- [i1] Homepage von Bruce Schneier - The Blowfish Encryption Algorithm
<http://www.schneier.com/blowfish.html>
- [i2] Wikipedia - Blowfish (cipher) (English)
[http://en.wikipedia.org/wiki/Blowfish_\(cipher\)](http://en.wikipedia.org/wiki/Blowfish_(cipher))
- [i3] Wikipedia - Blowfish (Deutsch)
<http://de.wikipedia.org/wiki/Blowfish>
- [i4] verschiedene Blowfish - Implementierungen
<http://www.schneier.com/blowfish-download.html>
- [i5] Die Boxes mit der hexadezimalen Darstellung der Zahl Pi
<http://www.schneier.com/code/constants.txt>