

Versuchsbericht zum Versuch M-23-h Verschlüsselungsverfahren

Markus Ganzenmüller

9. November 2002

Inhaltsverzeichnis

1	Versuchsbeschreibung	2
1.1	Aufgabenstellung	3
1.2	Kolloquium	3
2	RSA Verschlüsselungsverfahren	4
2.1	RSA Verschlüsselung	5
2.2	Grundlagen	6
2.3	Satz von Euler	6
2.4	Euklidischer Algorithmus	6
2.5	Schlüsselerzeugung	6
2.6	Verschlüsselung	7
3	DES Verschlüsselungsverfahren	9
3.1	Allgemeine Beschreibung	10
3.2	Funktionsweise	10
3.3	Vorteile	11
3.4	Nachteile	12
4	IDEA Verschlüsselungsverfahren	13
4.1	Allgemeine Beschreibung	14
4.2	Funktionsweise	14
4.2.1	Verschlüsselung	14
4.2.2	Entschlüsselung	15

4.3	Vorteile	15
4.4	Nachteile	15
5	Trends bei biometrischen Verfahren	16
5.1	Trends bei biometrischen Verfahren	17
6	Begriffserklärungen: Steganographie und digitale Unterschrift	18
6.1	Begriffserklärung: Steganographie	19
6.2	Begriffserklärung: digitale Unterschrift	19
7	Eigenes Verschlüsselungsverfahren	20
7.1	Eigener Verschlüsselungscode	21
7.1.1	main.c	21
7.1.2	coder.c	23
7.1.3	keygenerator.c	26

Kapitel 1

Versuchsbeschreibung

1.1 Aufgabenstellung

1. Entwickeln Sie ein einfaches Programm, das Daten verschlüsseln und entschlüsseln kann. Überlegen Sie sich dazu einen eigenen Algorithmus.
2. Erstellen Sie eine Art RSA-Code zur Verschlüsselung und Entschlüsselung der Daten. Sonderaufgabe: Ein Gruppenmitglied entwickelt ein einfaches Verschlüsselungsverfahren, das zweite Mitglied der Gruppe entwickelt ein Programm, das versucht, die Verschlüsselung zu knacken.
3. Testen Sie das Lernprogramm Interaktives Training Kryptologie von Hagemann/Hirse Korn (s.u.)
4. Testen Sie ein Programm, das den Rijndael-Algorithmus nach voll zieht. (evtl. aus dem Internet laden)
5. Testen Sie das Programm CrypTool (<http://www.CrypTool.de>)

1.2 Kolloquium

1. Erklären Sie die Grundlagen Ihrer Programmierung, das benutzte Verschlüsselungsverfahren, die Sicherheitsaspekte und die Unterschiede zwischen symmetrischen und asymmetrischen Verfahren.
2. Erklären Sie die Verfahren RSA, IDEA und DES.
3. Beschreiben Sie Stärken und Schwächen einzelner Verfahren und praktische Anwendungen wie z.B. Pretty Good Privacy. Beschreiben Sie Trends bei den biometrischen Verfahren.
4. Erklären Sie die Begriffe “Steganographie” und “digitale Unterschrift”.

Kapitel 2

RSA Verschlüsselungsverfahren

2.1 RSA Verschlüsselung

1976 forderten Whitfield Diffie und Martin Hellman eine neue Art der Verschlüsselungsalgorithmen, sogenannte asymmetrische Chiffrieralgorithmen oder auch public-key-Algorithmen genannt. Bei diesen Verfahren sollte es unmöglich sein mit Kenntnis des Chiffrierschlüssels den Dechiffrierschlüssel zu ermitteln. Die bisherigen symmetrischen Algorithmen hatten zwei wesentliche Nachteile:

1. Schlüsselmanagementproblem:

Jeder Teilnehmer muß mit jedem anderen Teilnehmer einen gemeinsamen Schlüssel vereinbaren. Dadurch werden zum Beispiel bei 1000 Anwendern 499.500 verschiedene Schlüssel notwendig (oder allgemein: bei n Anwendern $n(n-1)/2$ Schlüssel). Kommt ein neuer Teilnehmer hinzu, muß dieser mit jedem bisherigen Anwender einen eigenen Schlüssel vereinbaren. Dies bedeutet einen erheblichen Aufwand, sobald ein neuer Teilnehmer hinzukommt.

2. Symmetrie zwischen Chiffrier- und Dechiffrierverfahren:

Aus dem Chiffrierschlüssel kann man den Dechiffrierschlüssel ableiten. Dies kommt daher, daß beide Schlüssel entweder identisch oder sehr ähnlich sind. Fängt ein Dritter die chiffrierte Nachricht ab und kennt den Chiffrierschlüssel, kann er die Nachricht auch dechiffrieren.

Daraufhin entwickelten Ronald Rivest, Adi Shamir und Leonard Adleman 1977 das nach ihnen benannte public-key-Kryptosystem, den RSA-Algorithmus. Er erfüllt die von Diffie und Hellman geforderten Spezifikationen: Grundlegende Verschiedenheit zwischen Chiffrier- und Dechiffrierschlüssel Der Chiffrierschlüssel wird öffentlich bekannt gegeben, der Dechiffrierschlüssel bleibt geheim.

Dazu ein Beispiel:

Jeder Teilnehmer hat zwei Schlüssel: den Chiffrierschlüssel c und den Dechiffrierschlüssel d . d hält er geheim und c schreibt er in ein Telefonbuch eine Datei, die für jeden zugänglich ist und alle Chiffrierschlüssel c_n enthält. Möchte nun ein Teilnehmer A eine Nachricht an einen Teilnehmer B versenden, sucht er im Telefonbuch nach dem öffentlichen Schlüssel von B (c_B) und verschlüsselt damit seine Nachricht. Nur B kann dann mit seinem geheimen Dechiffrierschlüssel (d_B) diese wieder entschlüsseln. Kommt ein neuer Teilnehmer hinzu, wird nur ein neuer Schlüssel in das Telefonbuch eingetragen. Alle anderen Teilnehmer können sofort mit diesem kommunizieren, da ein Schlüsseltausch wie bei symmetrischen Verfahren nicht erforderlich ist. Es wird lediglich das Telefonbuch aktualisiert.

2.2 Grundlagen

2.3 Satz von Euler

Der Satz von Euler lautet:

Sind m und n zwei natürliche teilerfremde Zahlen, so gilt:

$$m^{\phi(n)} \text{ von } n \text{ MOD } n = 1;$$

$\phi(n)$ = Die Anzahl der der zu n teilerfremden natürlichen Zahlen

2.4 Euklidischer Algorithmus

Der Euklidische Algorithmus liefert den größten gemeinsamen Teiler (ggT) zweier natürlicher Zahlen. Er wird beim RSA - Algorithmus zur Berechnung des Codier- und Decodierschlüssels benötigt. Man erhält den ggT zweier natürlicher Zahlen a und b , indem man solange den Rest der Ganzzahlendivision ermittelt, bis der Rest 0 ist. x und y sind Puffervariablen und enthalten zu Anfang die Werte von a und b . Ist der Rest gleich 0, dann ist y der ggT. Solange der Rest ungleich 0 ist, wird x der Wert von y zugewiesen und y erhält den Wert von r .

2.5 Schlüsselerzeugung

Zuerst legt man zwei Primzahlen p und q fest, die für eine sichere Verschlüsselung circa in der Größenordnung 10200 liegen. Außerdem sollten sich die Stellenzahlen von p und q in der Dezimaldarstellung deutlich unterscheiden, da sich sonst beide Zahlen nur wenig von Wurzel n unterscheiden und durch gezielte Suchalgorithmen (z.B. mit dem Verfahren der Differenz der Quadrate) relativ leicht auffindbar wären.

Als nächstes bestimmt man das Produkt n der beiden Primzahlen und bildet $\phi(n)$, nach dem Satz von Euler.

Nun errechnet man schließlich die eigentlichen Schlüssel c (für Codier Schlüssel) und d (für Decodier Schlüssel) durch die Formel:

$$(c*d) \text{ MOD } \phi(n) = 1;$$

Hierzu wählt man den Schlüssel c so, daß c teilerfremd zu $\Phi(n)$ und daß der $\text{ggT}(c-1, p-1)$ und der $\text{ggT}(c-1, q-1)$ möglichst klein sind.

Um zu erreichen, daß c teilerfremd zu $\Phi(n)$ ist, kann man entweder für c eine Primzahl wählen, die größer als $\Phi(n)$ ist, oder man bestimmt zufällig eine Zahl c kleiner als $j(n)$, für die gilt: $\text{ggT}(c, j(n)) = 1$ (nachzuprüfen mit dem Euklidischen Algorithmus).

Daraus ergibt sich dann der Decodier Schlüssel d zu:

$$(c*d) \text{MOD } \Phi(n) = 1;$$

Da sich bei der Ganzzahldivision für das Produkt von c und d durch $\Phi(n)$ ein Rest von 1 ergibt, kann man das Produkt von c und d auch als eins plus ein ganzzahliges, positives Vielfaches von $j(n)$ schreiben:

$$\begin{aligned} c*d &= 1+k*\Phi(n) \quad k \text{ element von } \mathbb{N} \\ c*d &= 1+k*(p-1)*(q-1) \end{aligned}$$

Nun werden n und der Codier Schlüssel c veröffentlicht bzw. ins Telefonbuch eingetragen, d hält der Benutzer geheim und p und q werden vergessen. Es ist fast unmöglich, aus n $j(n)$ zu ermitteln ohne Kenntnis von p und q . Somit ist eine hohe Sicherheit der Verschlüsselung gewährleistet. Bei der Findung von n , c und d ist zu beachten:

1. Bei verschiedenen Benutzern ist darauf zu achten, daß nicht das selbe n mehrmals vorkommt, da man ansonsten nach dem Lemma von Bachet die Nachricht entschlüsseln kann, wenn sie an zwei Benutzer mit gleichem n geschickt wird.
2. Die $\text{ggT}(c-1, p-1)$ und $\text{ggT}(c-1, q-1)$ müssen möglichst klein sein, damit die Anzahl der Fixpunkte - dies sind die x von C , für die gilt: $xc = x \text{ MOD } n$, d.h. das ursprüngliche Zeichen entspricht dem verschlüsselten Zeichen - der Abbildung C möglichst klein ist. Denn je mehr Fixpunkte ein Verschlüsselungsalgorithmus besitzt, desto einfacher ist es, ihn zu entschlüsseln

2.6 Verschlüsselung

Da nur Zahlen verschlüsselt werden können, wird der zu verschlüsselnde Text t über den ASCII (American Standard Code for Information Interchange) Code in eine Zahlenfolge konvertiert. Zur Realisierung mittels PC werden diese Zahlen wiederum in einen Binärcode umgewandelt. Dieser wird dann in Blöcken der Reihe nach verschlüsselt (beim ASCII

Code sind die Blöcke zum Beispiel 512 Bit groß).
Die Verschlüsselung funktioniert dann folgendermaßen:

$$g = t^c \text{ MOD } n;$$

Der Geheimtext g ist der Rest der Ganzzahldivision aus dem ursprünglichen Text t potenziert mit dem Codier Schlüssel c des Adressaten und n .

SubsectionEntschlüsselung Die Entschlüsselung erfolgt dementsprechend einfach:

$$t = g^d \text{ MOD } n;$$

Der ursprüngliche Text t ist der Rest der Ganzzahldivision aus dem Geheimtext g potenziert mit seinem persönlichen geheimen Decodier Schlüssel d und n .

Dadurch, daß nur der Adressat über den richtigen Decodier Schlüssel verfügen kann, ist die Sicherheit der Verschlüsselung gewährleistet.

Kapitel 3

DES Verschlüsselungsverfahren

3.1 Allgemeine Beschreibung

DES bedeutet "Data Encryption Standard" (Datenverschlüsselungs-Standard). Es ist ein 1974 von IBM entwickeltes Verschlüsselungssystem, welches bis vor ein paar Jahren von der amerikanischen Regierung offiziell verwendet wurde. Ab 1976 wurde der Algorithmus von der NSA weiterentwickelt. IBM verzichtete im Nachhinein auf bereits angemeldete Patente.

DES basiert auf der symmetrischen Verschlüsselung je 64 Bit großer Blöcke mit einem 56-Bit-Schlüssel. Auch heute werden immer noch viele Dateien und Nachrichten mit dem DES-Algorithmus verschlüsselt, welcher jedoch nicht mehr als sicher gilt, da er, aufgrund der Leistungsstärke heutiger Computer, schon mehrere Male geknackt wurde. Es bedarf zwar immer noch eines großen Aufwands, dies zu bewerkstelligen, ist aber in wesentlich kürzerer Zeit möglich als noch vor ein paar Jahren.

3.2 Funktionsweise

Der Data Encryption Standard ist ein symmetrischer Algorithmus, der auf Blöcken der Größe 64 Bits arbeitet. Die Schlüssellänge beträgt ebenfalls 64 Bit, wobei jedoch 8 Bit davon als Paritätsbits verwendet werden, wodurch die effektive Schlüssellänge auf 56 Bit schrumpft.

Die Ver- bzw. Entschlüsselung eines 64-Bit-Blocks läßt sich grob in drei Schritte unterteilen:

1. Der Eingabeblock wird einer Eingangspermutation unterworfen, d.h. die Reihenfolge der Bits wird verändert. Das Ergebnis wird in zwei 32-Bit-Registern L und R gespeichert. Ähnlich wird mit den 64 Bit des Schlüssels verfahren. Es werden die vom Algorithmus benutzten 56 Schlüsselbits ausgewählt, ebenfalls permutiert und in zwei 28-Bit-Register C und D aufgeteilt.
2. In 16 Durchläufen werden 16 mal die gleichen Rechenschritte ausgeführt, wobei in jeder Runde auf den Werten von L und Operationen ausgeführt werden, die durch einen von 16 Teilschlüsseln beeinflusst werden. Das Ergebnis jeder Runde wird wieder in L und R zurückgeschrieben und in der nächsten Runde erneut verarbeitet.
3. Nach der letzten der 16 Runden werden L und R wieder zu einem 64-Bit-Wert zusammengefügt und invertiert.

Der Ablauf eines der 16 Verschlüsselungsschritte ist aus Abbildung 1 (siehe Beiblatt) ersichtlich. Zunächst wird der 32-Bit-Wert R durch eine Expansionsabbildung E auf einen 48-Bit-Wert erweitert. Hierbei bleibt die Reihenfolge der meisten Bits unverändert, es werden lediglich einige Bits verdoppelt. Das Ergebnis wird mit einem 48-Bit-Teilschlüssel XOR-verknüpft.

Jetzt werden die 48 Bits in acht Teilblöcke zu je 6 Bits zerlegt und für jeden der Blöcke wird mit Hilfe einer S-Box S_1 bis S_8 eine Substitution durchgeführt. Eine S-Box ist nichts weiter als eine Tabelle, die durch den Eingabewert indiziert wird. Die Tabelleneinträge sind konstant und durch den Standard definiert.

Da der Input jeder S-Box 6 Bits breit ist, der Output jedoch nur 4 Bits, beträgt die Breite des gesamten Blocks nach der Substitution wieder 32 Bit. Man kann die Funktion der S-Boxen deswegen auch dahingehend interpretieren, daß durch die redundanten Bits, die durch die Expansion erzeugt werden, für jede S-Box eine von vier möglichen bijektiven 4-Bit-Substitutionen angesteuert wird. Die S-Boxen gewährleisten aufgrund ihrer Nicht-Linearität mehr als alles andere die Sicherheit des Verschlüsselungsverfahrens.

Nach dem Aneinanderfügen der Ausgaben der S-Boxen entsteht wieder ein 32-Bit-Wert, welcher erneut permutiert wird. Das Ergebnis wird mit dem L-Register XOR-verknüpft und in das R-Register geschrieben. Der vorherige Inhalt von R wird auf L übertragen.

Die Vorgehensweise bei der Auswahl der Teilschlüssel wurde bisher nicht vollständig erklärt. Wie bereits erwähnt, werden die Bits des externen Schlüssels zunächst permutiert, und dann in zwei 28-Bit-Register C und D aufgeteilt. Diese Register werden nun vor jedem der 16 Durchläufe zyklisch um 1 oder 2 Bits nach links verschoben. Da die Anzahl der Schiebeoperationen über alle Runden hinweg 28 beträgt, enthalten C und D am Ende einer Ver- oder Entschlüsselungsoperation wieder den ursprünglichen Wert, so daß die Register für weitere Datenblöcke nicht neu geladen werden müssen. Um in jeder Runde einen Teilschlüssel für die XOR-Verknüpfung mit dem expandierten Register R zu gewinnen, ist eine weitere Permutation vonnöten. Sie wählt 48 Bits der 56 Bits aus und ändert deren Reihenfolge.

Nach dem letzten der 16 Durchläufe werden L und R wieder zu einem 64-Bit-Wert zusammengesetzt und die Reihenfolge der Bits invertiert. Danach kann der nächste Block verschlüsselt werden.

3.3 Vorteile

DES weist einige Besonderheiten auf. Da es sich um eine Blockchiffre handelt, die Klartextblöcke in Chiffretextblöcke gleicher Größe überführt, ist es nicht verwunderlich, daß sie bijektiv ist. Dagegen scheint es zunächst außergewöhnlich, daß der Algorithmus be-

züglich der bitweisen Invertierung invariant ist, d.h. $\text{DES}(K, M) = \text{not DES}(\text{not } K, \text{not } M)$. Dies ist darauf zurückzuführen, daß bei gleichzeitiger Invertierung des Datenblocks und des Teilschlüssels der Eingabewert für die S-Boxen gleich bleibt. Der Avalanche-Effekt wird von DES in hohem Maße erreicht. Bereits nach fünf Runden ist DES vollständig.

3.4 Nachteile

Ein mögliches Sicherheitsproblem stellen die schwachen Schlüssel des DES dar. Hierbei handelt es sich um externe Schlüssel, die dazu führen, daß nach der Anwendung der Permutationsfunktion die Register C und D jeweils nur gesetzte oder nur gelöschte Bits enthalten, so daß die 16 internen Teilschlüssel zwangsläufig alle identisch sind. Mit solchen Schlüsseln verschlüsselte Nachrichten sind erheblich leichter zu knacken. Neben den vier schwachen Schlüsseln existieren auch noch 12 semi-schwache Schlüssel, bei denen die internen Schlüssel nur zwei verschiedene Werte annehmen.

Mit Sicherheit der gravierendste Schwachpunkt des Algorithmus ist jedoch die geringe Mächtigkeit des Schlüsselraums von 2^{56} . Dadurch wird ein brute search nach dem richtigen Schlüssel, d.h. Durchprobieren aller Möglichkeiten, erlaubt, was die anderen positiven Eigenschaften der Chiffre zunichte macht. Da es die NSA war, die die 128 Schlüsselbits des ursprünglichen Vorschlags von IBM auf nur 56 Bits verringerte, kann spekuliert werden, daß dieser Effekt erwünscht ist.

Zusammenfassend kann man sagen, daß die Mängel des DES mittlerweile so schwerwiegend sind, daß das Verfahren nicht mehr als sicher gelten kann. Auch das Ausweichen auf Triple-DES stellt nur eine Notlösung dar, weil bei seiner Verwendung Geschwindigkeitseinbußen in Kauf genommen werden müssen.

Kapitel 4

IDEA Verschlüsselungsverfahren

4.1 Allgemeine Beschreibung

Als Alternative zum DES entwickelten Xuejia Lai und James L. Massey eine Blockchiffre, die neben einer höheren Sicherheit, vor allem durch einen größeren Schlüsselraum, auch schnellere Implementationen in Hardware und Software bot. Der Algorithmus wurde 1990 als “Proposed Encryption Standard” (PES) vorgestellt, es zeigte sich jedoch, daß er gegen die kurz darauf veröffentlichte differentielle Kryptoanalyse nicht immun war, so daß der Algorithmus geringfügig verändert wurde. Diese Variante bezeichnet man heute als “Improved” DES (IPES) oder IDEA.

IDEA arbeitet genauso wie DES auf 64-Bit-Datenblöcken, wobei die Schlüssellänge auf 128 Bit erweitert wurde. Eine Verschlüsselungsoperation besteht aus acht Durchläufen gefolgt von einer Ausgabetransformation. Um die Bedingung von schneller Implementierbarkeit in Software zu erfüllen, wurde auf Permutationen verzichtet. Die Grundoperationen beschränken sich auf XOR, Addition modulo 2^{16} , sowie Multiplikationen modulo $2^{16} + 1$, wobei der Operand 0 als 2^{16} interpretiert wird.

4.2 Funktionsweise

4.2.1 Verschlüsselung

Die Funktionsweise von IDEA ist in Abbildung 2 (siehe Beiblatt) für die erste Runde und die abschließende Ausgabetransformation dargestellt. Der 64 Bit breite Klartextblock M wird zunächst in vier Teilblöcke M_1 bis M_4 zu je 16 Bits aufgeteilt. In jeder Runde $r = 1 \dots 8$ werden nun beeinflusst von sechs Teilschlüsseln $K_{r,1}$ bis $K_{r,6}$ verschiedene Operationen durchgeführt, deren Ergebnis nach einer Permutation der Teilblöcke von der nächsten Runde weiterverarbeitet wird.

Insgesamt werden sechs Schlüssel für jede der acht Runden zuzüglich weiterer vier für die Ausgabetransformation benötigt, also zusammen 52 Teilschlüssel zu je 6 Bits. Jeweils vier der Teilschlüssel werden aus dem 128-Bit-Schlüssel generiert, indem dieser in acht 16-Bit-Blöcke aufgeteilt wird. Nachdem der 128-Bit-Schlüssel zyklisch um 25 Bits nach links verschoben wurde, steht er dazu bereit für die nächsten vier Teilschlüssel aufgeteilt zu werden.

4.2.2 Entschlüsselung

Die Entschlüsselung gestaltet sich größtenteils analog zur Verschlüsselung, die Teilschlüssel müssen lediglich in umgekehrter Reihenfolge verwendet werden. Darüber hinaus werden einige der Teilschlüssel verändert: Der in Abbildung 2 gestrichelt umrandete Teil der Operationen ist selbstinvers, so daß $K_{r,5}$ bis $K_{r,6}$ unverändert bleiben können. Für $r = 1..8$ müssen jedoch $K_{r,1}$ sowie $K_{r,4}$ bezüglich der Multiplikation modulo $2^{16} + 1$ und außerdem $K_{r,2}$ sowie $K_{r,3}$ bezüglich der Addition modulo 2^{16} invertiert werden. Es ist stets gewährleistet, daß für die Multiplikation ein Inverses existiert, da es sich bei $2^{16} + 1$ um eine Primzahl handelt.

4.3 Vorteile

Neben seiner guten Implementierbarkeit hat der IDEA-Algorithmus den Vorteil des großen Schlüsselraumes von 2^{128} Bit, welcher heute als Standard gilt. Da es sich um eine symmetrische Verschlüsselung handelt, ergeben sich auch bei der Verschlüsselungszeit Vorteile. Außer den unten beschriebenen, wurden bisher keine Schwächen entdeckt, was die Qualität des Algorithmus beweist.

4.4 Nachteile

Beim IDEA existieren ähnlich wie beim DES schwache Schlüssel, die sich dadurch auszeichnen, daß interne Teilschlüssel den Wert Null oder Eins annehmen. Null zeigt bei der Addition und XOR-Verknüpfung keine Wirkung, bei der Multiplikation ist Null selbstinvers, da laut Konvention $0 = 2^{16} = -1$. Ein Teilschlüssel mit dem Wert 1 hat bei der Multiplikation keinen Effekt. Dies bedeutet v.a., daß die Anwendung von IDEA mit einem externen Schlüssel, bei dem alle Bits gelöscht sind, selbstinvers ist. Es existieren weitere schwache Schlüssel, die einen Angriff auf den Algorithmus ermöglichen. Andere Schwächen wurden in der Chiffre nicht gefunden.

Kapitel 5

Trends bei biometrischen Verfahren

5.1 Trends bei biometrischen Verfahren

Der Einsatz biometrischer Identifikationsverfahren, zum Beispiel für die innere Sicherheit gewinnt zunehmend an Bedeutung, da sie eine eindeutige Identifikation von Personen ermöglichen. Biometrische Verfahren werden insbesondere an Flughäfen, bei Grenzkontrollen, als Zutrittskontrolle für Sicherheitsbereiche sowie bei Informationstechnologien verwendet.

Eine weitere Einsatzmöglichkeit besteht im Rechtsverkehr: Für die Freischaltung des privaten Schlüssels zur Erzeugung der Signatur kann ein biometrisches Merkmal anstatt oder zusätzlich zu einer PIN zur Identifikation des Besitzers der Chipkarte eingesetzt werden.

Alle biometrischen Verfahren sind nach dem gleichen Grundprinzip aufgebaut: Am Anfang steht die Personalisierung, das heißt, unverwechselbare körpereigene Merkmale, wie Fingerabdrücke, Iris und Netzhaut, aber auch die ganze Hand, Gesicht, Lippen oder Stimme werden vermessen. Diese Messdaten werden allerdings nicht komplett gespeichert. Stattdessen werden charakteristische Merkmale extrahiert. Dies geschieht mit Hilfe eines Merkmals-Extraktionsalgorithmus. Die gewonnenen Daten werden verschlüsselt und in so genannten Templates hinterlegt. Das kann zum Beispiel eine Chipkarte oder eine Datenbank sein. Bei jedem Identifikationsprozess werden die biometrischen Merkmale erneut von einem Sensor aufgenommen, die charakteristischen Daten extrahiert und mit dem gespeicherten Template der zu identifizierenden Person verglichen. Sind beide Datensätze identisch, wird der Zugriff auf die verlangten Daten, bzw. der Zutritt zum Sicherheitsbereich gewährt, usw....

Bisher sind biometrische Verfahren allenfalls als zusätzliche Absicherung zu Standardverfahren sinnvoll, da biometrische Erkennungssysteme mit teils geringem Aufwand getäuscht werden können. Die Zeitschrift c't unternahm vor ein paar Monaten den Versuch, biometrische Systeme zu täuschen, um sich so Zugang zu geheimen Daten zu verschaffen. Mit großem Erfolg. Ein System zur Gesichtserkennung ließ sich z.B. durch das Foto der entsprechenden Person täuschen, welches auf einem TFT-Bildschirm vor die Kamera gehalten wurde. Ferner war es möglich, Fingerabdrucksensoren dadurch zu täuschen, dass die Fingerabdrucke einer Person von einem Trinkglas extrahiert und auf ein Stückchen Gelee übertragen wurden. Dieses wurde leicht erwärmt, da der Abdruckssensor zusätzlich einen Wärmesensor enthielt. Das Geleestück wurde nun auf den Sensor gelegt. Auch dieses Mal bekam man Zutritt zum System.

Diese Untersuchung hat wesentlich dazu beigetragen, dass sich der "Hype" um biometrische Erkennungssysteme gelegt hat und sich die Entwickler zurückgezogen haben um ihre Systeme zu überarbeiten und zu verbessern.

Kapitel 6

Begriffserklärungen: Steganographie und digitale Unterschrift

6.1 Begriffserklärung: Steganographie

Ein steganographisches Verfahren verheimlicht, daß überhaupt geheime Daten existieren. Der Gedanke dahinter: Wo niemand geheime Daten vermutet, wird sie auch niemand suchen. Steganographie-Software versteckt die geheime Datei in einem anderen Dokument. Als sogenannte Trägerdateien dienen in der Regel meist Bilder, Sound-, Video- oder Textdateien. Dieses Verfahren kann nicht nur zum Schutz von Daten benutzt werden sondern wird auch zur Kenntlichmachung von Urheberrechten verwendet.

Wer von der Verschlüsselung nichts weiß, nutzt die betreffende Trägerdatei ohne Einschränkungen mit der passenden Anwendung. Nur wer über die Verschlüsselung informiert ist und zudem den Zugriff auf den verwendeten Kodierungs-Schlüssel hat, kann die in der Trägerdatei enthaltenen Informationen entschlüsseln und für sich nutzbar machen.

6.2 Begriffserklärung: digitale Unterschrift

Die elektronische Unterschrift ist ein Versuch, sich auch im anonymen Internet eindeutig identifizieren zu können. Das ist vor allem gefragt bei Online-Einkäufen und beim Versenden von E-Mails mit sensiblem Inhalt. Noch gilt die elektronische Unterschrift nicht überall als rechtlich anerkanntes Pendant zur "echten" Unterschrift einer Person. Doch in der Praxis werden im Internet bereits Verfahren wie PGP (Pretty Good Privacy) angewendet und akzeptiert, die digitale Signaturen eindeutig einem Absender zuweisen:

- Um eine E-Mail digital zu unterschreiben, benötigt jeder Absender einen öffentlichen und einen privaten Schlüssel: Public und Private Key. (Das können dieselben Schlüssel sein, mit denen der Inhalt einer E-Mail verschlüsselt wird - also für fremde unleserlich gemacht wird.)
- Beim Verschicken einer E-Mail signiert der Absender diese mit seinem privaten Schlüssel, dem elektronischen Pendant zur "echten" Unterschrift.
- Der Empfänger prüft nun die Echtheit des Absenders, indem er die Signatur mit dem öffentlichen Schlüssel des Absenders decodiert. Läßt sich die Unterschrift so verifizieren, ist die Nachricht eindeutig vom Absender und außerdem nach dem Signieren nicht mehr nachträglich geändert worden.

Kapitel 7

Eigenes Verschlüsselungsverfahren

7.1 Eigener Verschlüsselungscode

Nachfolgend der Code für eine Verschlüsselung nach dem Verfahren von Ceasar.

Um die Verschlüsselung nicht zu schnell nachvollziehen zu können haben wir noch eine kleine Abwandlung vorgenommen.

Das Verfahren nach Caesar beruht darauf dem zu verschlüsselnden Zeichen einen Ascii Wert zu addieren. Das Programm berechnet eine pos. Zahl als Product zweier vom Benutzer eingegebenen Zahlen und fügt 1000 hinzu.

7.1.1 main.c

```
1  /* Montag Oktober 2002*/
2  /* FOR BEST FORMATING OF THIS FILE SET TABSTOP TO 2 */
3
4  #include <stdio.h>           /* because of file functions */
5  #include <string.h>         /* because of strcmp */
6  #include "keygenerator.h"   /* generates keys to chiffre or
7                               dechiffre*/
8  #include "coder.h"         /* encodes or decodes */
9
10 int CheckParsed( char* choice, char* filename );
11
12 /*
13 *Funktion:                main
14 *Description            this routine collects information
15                          and calls the encoder/decoder sequence
16 *Parameters
17 *Returnvalue           0 for success, non-zero on failure
18 */
19
20 char*    programname;
21 int      tocode; // = 1; encode by default, 0 = decode
22 FILE*    file;           // file 2de/encode
23
24
25 int main( int argc, char** argv )
26 {
27
28 int encodekey, decodekey;
```

```

29  int n;
30
31      char*    filename2de_or_encode,*choice;
32  //,*string;    // filename of file to decode/encode(argv[3])
33  // string to be read from stdin
34  // choice (decode or encode)(argv[2])
35
36  int    ret_val = 0;// return value of code funktion
37
38  programname = argv[0];
39
40  // parse the parameters
41  if( argc == 3)
42  {
43  choice = argv[1];    //decides whether to de/en code
44  filename2de_or_encode = argv[2];
45  if ( CheckParsed( choice, filename2de_or_encode )!= 0 )
46      return(1);
47  }
48  else
49  { fprintf(stderr,"usage: %s encode|decode [filename]\n",*argv);
50    return (0);
51  }
52
53
54
55      //call the working funktion
56  if( tocode )
57  {
58  GenerateKey( &encodekey, &decodekey, &n );
59  ret_val = FileEncode(file, &encodekey,
60  &decodekey,filename2de_or_encode );
61  }
62  else
63  {
64  printf( " I see, you know this program :-)
65  \n Please type your decodekey and press 'enter\n");
66  fflush(stdin);
67  scanf("%d",&decodekey);
68  ret_val = DeCoding(file,filename2de_or_encode,&decodekey);
69  }

```

```

70
71
72 return ( ret_val );
73 }
74
75 int CheckParsed( char* choice, char* filename )
76 {
77 //check input parameters
78 if( strcmp (choice, "decode" ) == 0 )
79     tocode = 0;
80 else if( strcmp (choice, "encode" ) == 0)
81     tocode = 1;
82 else
83 {
84     fprintf(stderr,"usage: %s encode|decode
85             [filename]\n",programname);
86     return (1);
87 }
88
89 if( !(file = fopen( filename, "r" ) ) )
90 {
91     fprintf( stderr, "error: can't open file\n %s",filename );
92     return (1);
93 }
94
95 return (0);
96 }

```

7.1.2 coder.c

```

1  #include <stdio.h>
2  #include <math.h>
3  #include <assert.h>
4  #include "coder.h"
5
6
7  // Globals
8  int* codekey; int* dcodekey;
9  FILE* newfile;
10 /*

```

```

11  *Funktion:                FileEnCode
12  *Description            this routine reads a file, one char another,
13                          and call the routine EnCoding to encode the chars
14  *Parameters            file pinter to read from,generated
15                          encodekey/decodekey,
16                          filename of the file
17  *Returnvalue          0 for success, non-zero on failure
18  */
19
20  int FileEncode(FILE* file,int* encodekey,
21                int*decodekey, char * filename)
22  {
23  char buffer[1];
24  int ascii,bstabe;
25  assert(file!=0);
26  dcodekey = decodekey;
27  codekey = encodekey;
28  newfile=fopen("encodedfile.txt","w");
29
30  while( fread(buffer,sizeof(char),1,file)!= 0 )
31  {
32
33  ascii = (int)buffer[0];
34  bstabe = ascii;
35
36  EnCoding(bstabe);
37
38  }
39  fclose(newfile);
40  filename = "encodedfile.txt";
41
42  return (0);
43  }
44  /*
45  *Funktion:                EnCoding
46  *Description            this routine encodes the give
47  char as int to write it in a new file called encodedfile.txt
48  *Parameters
49  *Returnvalue          0 for success, non-zero on failure
50  */
51  int EnCoding(int ascii)

```

```

52  {
53
54  unsigned long int newcodeword;
55
56  newcodeword = (unsigned long int) ascii + *codekey;
57  fwrite(&newcodeword,sizeof(unsigned long int),1,newfile);
58
59  return 0;
60  }
61
62  /*
63  *Funktion:          main
64  *Description      this routine collects information and calls
65                   the encoder/decoder sequence
66  *Parameters
67  *Returnvalue     0 for success, non-zero on failure
68  */
69
70  int DeCoding(FILE* file, char* filename,int* decodekey)
71  {
72  unsigned long int  buffer [1];
73  int ascii;
74
75  FILE* decodedfile;
76
77  fopen(filename,"r");
78  decodedfile = fopen("decodedfile.txt","w");
79
80  while ( fread(buffer,sizeof(unsigned long int),1,file) )
81  {
82      ascii = buffer[0];
83      ascii = (unsigned long int )ascii - *decodekey;
84      buffer[0] = (char) ascii;
85      {
86      char c = (char)buffer[0];
87      fwrite( &c,sizeof(char),1,decodedfile);
88      }
89
90  }
91  fclose(file);
92  fclose(decodedfile);

```

```

93 return (0);
94
95 }
96

```

7.1.3 keygenerator.c

```

1  #include <stdio.h>          /* because of file functions */
2  #include <math.h>          /*because of mathematics functions
3  #include "keygenerator.h"
4
5  /* DESCRIPTION
6   * Function      // generates key to en/decode
7                   and gives inrmation about the
8                   program through stdout
9                   // users information based on
10                  stdin are required
11   * Parameters   // none
12   * Return value // always 0
13  */
14
15
16
17 int GenerateKey(int* encodekey,int* decodekey,int* n)
18 {
19
20 /*declaration of variables*/
21 int p,q;                //to manage users information
22 //unsigned long int* n; //as result of p*q/
23                          2.part of the publickey
24
25 unsigned long int euler;// as result
26                          for prims after Mr.Euler (p-1)(p-1)
27
28
29
30 printf( " _____
31          _____\n");
32 printf( "*****\n");
33          *****\n");

```

```

34 printf( "This Program encodes and decodes files
35         using symetric algrithms\n");
36 printf( "\nRemember you need to decode and encode
37 files the same key, so keep in mind or secure place!\n");
38
39
40 printf( "*****
41                                             *****\n");
42 printf( "To generate a new key, there are some information
43                                             to be added
44 -----
45                                             -----\n" );
46 printf( "Please type two two numbers between 2 and 5000
47         each one followed by a \"return\"\\n");
48 scanf ( "%d",&p);
49 fflush( stdin ); //flushes the stdin buffer to make sure
50                 that its empty when reading the next prime
51 scanf ( "%d",&q);
52 fflush( stdin );
53
54
55 /* some maths to arithmetic the key, describes ceasars
56 method to shift the chars in one way,with one number*/
57 *n = p * q;
58 euler = (p-1)*(q-1);
59 *encodekey = (1000 + euler);
60 *decodekey = *encodekey;
61 /*printing stdout full keys information*/
62 printf("*****\n" );
63 printf("This is your de\\encode key :\n {%d}\n",
64        (int)*encodekey);
65
66
67
68 return (0);
69 }
70
71
72
73 /*DESCRIPTION
74 * Function // Checks if parameter is a prime,

```

```

75                                     if not it calls itself
76  * Parameter      // number to be checked as an int value
77  * Return value  // a correct prime
78  */
79  /*int CheckPrime(int x)
80  {
81  int n = 2;
82  printf("x: %d \n", (int)x);
83  for( ; n < x; n++)
84  {
85  if(x % n == 0)
86  {
87  printf("n: %d \n", n);
88  printf( "Sorry, no Prime, please try a new one\n" );
89  fflush( stdin );
90  scanf("%d",&x);
91  fflush( stdin );
92  x = CheckPrime(x);
93  return (x);
94  }
95  }
96  return (x);
97  }*/
98
99

```