Sicheres Verschlüsseln mit asymmetrischen Verschlüsselungsalgorithmen

---

Patrick Reichert, Christoph Seidler Georg- Cantor- Gymnasium Halle (Saale)

Kurzfassung :

Kryptologie, die Lehre vom Verschlüsseln und Entschlüsseln, galt lange Zeit lediglich als Domäne des Militärs. Im Zeitalter der globalen Kommunikation werden Verschlüsselungsverfahren aber auch für Privatpersonen immer interessanter, da das Interesse an einer gesicherten Kommunikation stetig wächst. Für den Einsatz in solchen globalen Kommunikationssytemen sind asymmetrische Verschlüsselungsalgorithmen wegen ihrer einfachen Schlüsselverwaltung bei einem hohen Maß an Sicherheit von großem Vorteil. Bei diesen Verfahren benutzt man zum Verschlüsseln und zum Entschlüsseln zwei verschiedene Schlüssel.

Wir haben im Rahmen unserer Arbeit ein solches Ver- und Entschlüsselungssystem, basierend auf dem RSA- Algorithmus, in einem Programm für die Oberfläche MS Windows™ implementiert. Das Programm wurde im Hinblick auf die Geschwindigkeit beim Ver- und Entschlüsseln optimiert. Außerdem werden die verschlüsselten Nachrichten mit einer digitalen Unterschrift des Absenders versehen, was dem Empfänger eine Authentizitätsprüfung ermöglicht, die ebenfalls im Programm eingebunden ist.

In der vorliegenden Arbeit werden detailliert die mathematischen Funktionsweisen von asymmetrischen Verschlüsselungsalgorithmen erklärt und am Beispiel des RSA- Algorithmus demonstriert, außerdem wird das Programm und seine Funktionsweise ausführlich dokumentiert. Ebenso wird auf die theoretischen Grundlagen der digitalen Nachrichtensignaturen eingegangen.

Inhalt:

1 EINLEITUNG 3

2 ASYMMETRISCHE VERSCHLÜSSELUNGSALGORITHMEN 3

2.1 DER RSA- ALGORITHMUS 5

3 DIE SOFTWARE 6

3.1 PROGRAMMOBERFLÄCHE 7

3.1.1 Die Menüleiste 8

3.1.2 Die Hilfsmittelleiste 9

3.1.3 Der Textbereich 9

3.1.4 Die Listbox mit vorhandenen „public- keys“ 9

3.1.5 Die Statuszeile 9

3.2 STRUKTUR DES PROGRAMMES 9

4 IMPLEMENTIERUNG UND OPTIMIERUNG DER BERECHNUNGEN 10

4.1 ADDITION UND SUBTRAKTION VON DUALZAHLEN 11

4.2 MULTIPLIKATION EINER DUALZAHL MIT EINEM WORD (FUNKTION DZMALWORD) 11

4.3 MULTIPLIKATION ZWEIER DUALZAHLEN 12

4.4 BERECHNUNG DER KONGRUENZ (A % B) 12

4.5 MODULARE EXPONENTATION 13

5 ANHANG 14

5.1 EINDEUTIGKEIT DER VER- UND ENTSCHLÜSSELUNG 14

5.2 LITERATUR 16

5.3 DANKSAGUNGEN 16

1 Einleitung

Die moderne Kommunikation über die internationalen Datennetze ist für Privatpersonen, aber auch für Firmen mit einem erheblichen Sicherheitsdefizit behaftet, da nicht kontrolliert werden kann, ob Nachrichten nicht auf dem Weg vom Absender zum Empfänger von Dritten gelesen oder sogar manipuliert werden. Technisch ist dies ohne weiteres möglich und stellt deshalb, insbesondere bei sicherheitssensitiven Daten, wie Kranken- oder Finanzdaten, aber auch bei betrieblichen Daten, ein unkalkulierbares Risiko dar.

Zu diesem Zwecke werden kryptographische Verfahren zur Verschlüsselung der Daten genutzt [1]. Aufgrund von deutlichen Vorteilen auf dem Gebiet der Schlüsselverteilung und der Schlüsselanzahl werden hierfür sogenannte „asymmetrische Verschlüsselungssysteme“ (siehe auch 2.), oder auch „public- key“ - Verschlüsselungssysteme genutzt.

Da sich Endanwenderlösungen von Implementierungen der „public- key“- Systeme meist als sehr langsam und oft auch unkomfortabel erwiesen, beschlossen wir, eine eigene Implementierung eines asymmetrischen Kryptosystems in einem PC- Programm für die Windows™- Oberfläche vorzunehmen, die im Hinblick auf Geschwindigkeit und im besonderen Komfortabilität Vorteile bringt, die nicht mit Abstrichen an der Sicherheit bezahlt werden sollten. Das Programm und seine Funktionsprinzipien inklusive der mathematischen Grundlagen sind in der folgenden Arbeit dargelegt.

2 Asymmetrische Verschlüsselungsalgorithmen

Herkömmliche, sogenannte symmetrische Verschlüsselungssyteme sind zum einen dadurch gekennzeichnet, daß jeweils zwei Kommmunikationspartner einen gemeinsamen geheimen Schlüssel benutzen, zum anderen dadurch, daß ein sicherer Kanal für die Übertragung dieses Schlüssels gefunden werden muß.

Insbesondere bei einer größeren Anzahl an Kommunikationsteilnehmern wird die Schlüsselverwaltung durch die große Schlüsselanzahl unhandlich, da bei n Kommunikationsteilnehmern n*(n-1)/2 Schlüssel benötigt werden. Außerdem kann ab einer bestimmten Anzahl von Schlüsseln nicht mehr für die Sicherheit der Übertragungskanäle gesorgt werden, was wiederum einen Mangel an Kommunikationssicherheit nach sich zieht.

Eine großer Schritt zur Beseitigung dieser Mängel stellt die 1976 von Whitfield Diffie und Martin Hellman veröffentlichte Arbeit „New Directions in Cryptography“ [2]. In dieser Arbeit wird erstmals ein asymmetrisches Kryptosystem vorgestellt.

[1] Bereits Caesar nutzte ein Verschlüsselungsverfahren zur Sicherung wichtiger Daten, die Substitutionschiffre.

[2] siehe hierzu [4]

Danach gelten in einem beliebigen asymmetrischen Verschlüsselungssystem folgende Prinzipien:

- Jeder Kommunikationsteilnehmer *T* befindet sich im Besitz von **zwei** Schlüsseln :

1. einem **öffentlichen Schlüssel** E_T zur Verschlüsselung sowie
2. einem **geheimen Schlüssel** D_T zur Entschlüsselung.

- Alle Teilnehmer in einem asymmetrischen Kommunikationssystem verwenden denselben Verschlüsselungsalgorithmus *e* und denselben Entschlüsselungsalgorithmus *d*.

Für jede Nachricht *m* in einem asymmetrischen Verschlüsselungssystem gilt nun:

$$\boldsymbol{d}(\boldsymbol{e}(m,E_T),D_T) = m$$

Der Nachrichtenaustausch zwischen zwei Teilnehmern eines asymmetrischen Kryptosytems läuft dann wie folgt ab:

1. Der Benutzer *A* muß sich den öffentlichen Schlüssel E_B des Teilnehmers *B* aus einer öffentlichen Datei heraussuchen.

2. Der Benutzer *A* verschlüsselt das Kryptogramm mit Hilfe des Verschlüsselungsalgorithmus ***e*** und des öffentlichen Schlüssels E_B des Teilnehmers *B* und sendet das Kryptogramm

$$c = e(m,E_B)$$

über einen beliebigen offenen (also durchaus unsicheren) Kanal an *B.*

Benutzer *B* entschlüsselt nun das erhaltene Kryptogramm c mit dem Entschlüsselungsalgorithmus *d* und seinem geheimen Schlüssel D_B.

$$m = \boldsymbol{d}(c, D_B)$$

Zusammenfassend kann man sagen, daß ein asymmetrisches Kryptosystem folgenden Anforderungen genügen sollte:

1. Bei gegebenem m und E_T sollte $c = \boldsymbol{e}(m,E_T)$ leicht berechenbar sein.
2. Bei gegebenem Kryptogramm *c* sollte es rechnerisch nicht möglich sein, die Nachricht *m* zu erschließen, dies bedeutet konkret, daß aus Kenntnis des öffentlichen Schlüssels E_T der geheime Schlüssel D_T nicht zu erschließen sein darf.
3. Wenn das Kryptogramm *c* und der geheime Schlüssel D_T bekannt sind, sollte die Nachricht *m* leicht wieder zu bestimmen sein.

2.1 Der RSA- Algorithmus

Diffie und Hellmann hatten lediglich die theoretische Idee für eine asymmetrisches Kryptosystem geliefert, aber keine praktische Implementierung eines „public- key“ Kryptosystems aufgezeigt, dies blieb drei anderen Mathematikern vorbehalten: Ronald Rivest, Adi Shamir und Leonard Adlemann.

Mit dem 1978 veröffentlichten und am 20. September 1983 als US- Patent 4405829[3] eingetragenen RSA- Algorithmus lieferten sie die wohl wichtigste und bekannteste Implementierung eines asymmetrischen Kryptosystems. Seine Sicherheit hängt im Wesentlichen von der Vermutung ab, daß es keine schnelle Möglichkeit gibt, Zahlen zu faktorisieren, die das Produkt zweier hinreichend „großer“ Primzahlen sind. Konkret heißt dies, daß es mit den heutigen mathematischen und technischen Mitteln nicht möglich ist, eine Zahl mit deutlich mehr als 100 Dezimalstellen in angemessener Zeit in ihre Primfaktoren zu zerlegen.

Im folgenden sollen kurz die Verfahren zur Ver- beziehungsweise Entschlüsselung beschreiben werden:

1. Man bestimme zwei hinreichend „große“ Primzahlen *p* und *q* („groß“ bedeutet im Falle des RSA-Algorithmus meistens 200 stellig). Die Zahl *n* sei durch

   $$n = pq \text{ definiert.}$$

2. Man bestimme eine „große“ ganze Zufallszahl *d*, wobei gelte :

   $$ggT((p-1)(q-1), d) = 1 \text{ (d sei zur Zahl (p-1) (q-1) teilerfremd)}$$

3. Man berechne eine eindeutig bestimmte ganze Zahl *e* nach der Formel

   $$ed = 1 \ (\text{MOD} \ (p-1)(q-1)),$$

wobei gelte:

$$e \pounds (p-1)(q-1)$$

4. Man gebe den öffentlichen Schlüssel („public- key“) bekannt, der aus dem Paar ganzer Zahlen (*e*,*n*) besteht.

5. Man trenne die zu übertragende Nachricht *M* in gleich große Blöcke und stelle diese Blöcke als Zahl dar, wobei gelte :

   $$1 \pounds M \pounds n$$

6. Man verschlüssele *M* in das Kryptogramm *C*, wobei hier gelte :

   $$C = M^e \ (MOD \ n)$$

7. Eine Entschlüsselung dieses Kryptogramms *C* ist mit Hilfe des Privatschlüssels *d* möglich. Die entschlüsselte Nachricht berechnet sich demzufolge wie folgt:

   $$D = C^d \ (MOD \ n)$$

[3] siehe hierzu auch [7]

3 Die digitale Signatur

Die Authentizität von Nachrichten wird normalerweise durch eine Unterschrift bestätigt. Auch bei der Kommunikation über elektronische Post wäre eine Unterschrift wünschenswert, da Manipuationen am Text anderenfalls nicht vom Empfänger bemerkt werden könnten. Dieser Zustand ist insbesondere für geschäftliche Kommunikation untragbar, deswegen macht sich eine sogenannte digitale Signatur nötig. Wesentliche Forderungen an eine digitale Unterschrift, die mit kryptologischen Methoden erzeugt werden kann, sind:

1. Nur der rechtmäßige Absender eines Dokuments kann die Unterschrift erzeugen.
2. Der Empfänger des Dokuments kann die Unterschrift zweifelsfrei prüfen.
3. Die Unterschrift gilt nur im Zusammenhang mit dem gegebenen Dokument.

Asymmetrische Verschlüsselungsverfahren ermöglichen durch ihre Theorie eine technisch relativ einfach gehaltene Implementation der digitalen Signatur. Folgende Schritte sind für eine digitale Signatur nötig:

1. Der Absender komprimiert die Nachricht auf eine Art „Fingerabdruck“ F, der in Form einer Zahl vorgegebener Maximallänge angegeben wird. Diese Funktion kann nicht umkehrbar sein (andernfalls wäre sie eine ideale Kompressionsfunktion, die beliebig lange Nachrichten um einen vorgegebenen Faktor komprimiert). Diese Funktion ist also mit einem Informationsverlust verbunden; aus dem Ergebnis kann in keinem Fall auf die vollständige Nachricht geschlossen werden. (Einweg - Hash - Funktion)
2. Der Absender A verschlüsselt den „Fingerabdruck“ mit seinem (nur ihm bekannten) privaten Schlüssel D_A und veröffentlicht den signierten „Fingerabdruck“ $D_A(F)$.
3. Der Empfänger B kann nun die Signatur verifizieren, indem er zuerst selbst einen Fingerabdruck F´ der Nachricht berechnet und dann mit dem öffentlichen Schlüssel E_A vom Absender überprüft, ob folgende Gleichheit gilt: $E_A(D_A(F)) = F´$.

4 Die Software

Nachdem nun die theoretischen Grundlagen asymmetrischer Verschlüsselungssysteme, speziell des RSA- Algorithmus dargelegt worden sind, soll dieses Kapitel die programmiertechnische Umsetzung des theoretischen Wissens in Form eines Windows™- Programmes beschreiben. Für die Entwicklung des Programmes nutzten wir die Programmiersprache C++ der Firma Borland in der Version 4.5. Als Zielplattform wählten wir die Windows™- Umgebung, da hier ein ansprechendes Äußeres des Programmes realisiert werden kann, und dieses mit einer intuitiven Bedienfähigkeit einhergeht.

4.1 Programmoberfläche

Im folgenden soll die Benutzeroberfläche des Programms näher vorgestellt werden :

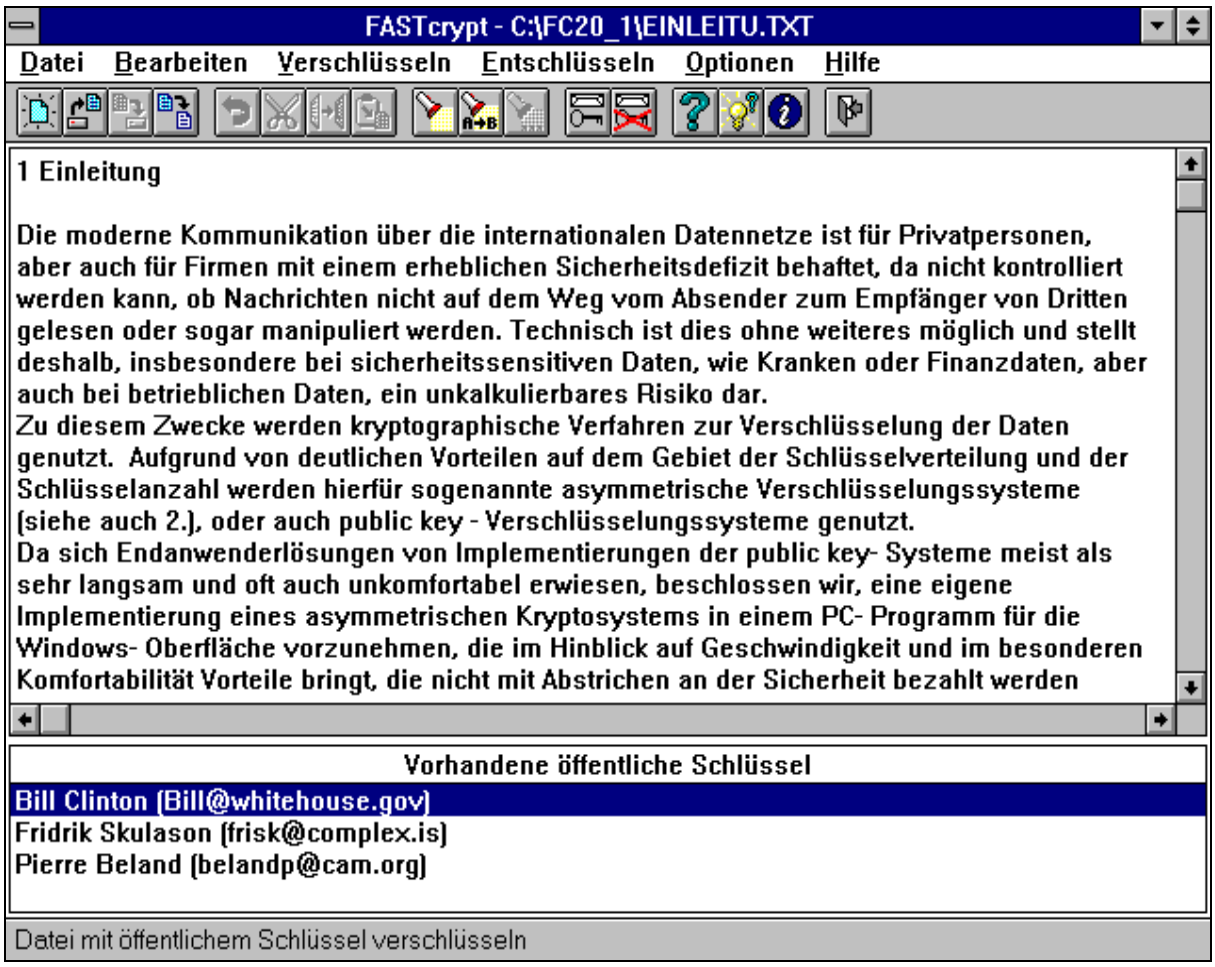

Abbildung 1: Die Programmoberfläche

Die Programmoberfläche gliedert sich in 5 visuelle Einheiten:

1. Menüleiste
2. Hilfsmittelleiste
3. Textbereich
4. Listbox mit vorhandenen „public- keys“
5. Statuszeile

4.1.1 Die Menüleiste

Diese Zeile erlaubt die Programmsteuerung mit Hilfe von Pop-up Menüs.
Der Menüpunkt **`Datei`** erlaubt:

- das **Anlegen** einer neuen Datei
- das **Öffnen** von Dateien
- das **Speichern** von Dateien
- das **Speichern** von Dateien unter einem anderen Dateinamen
- sowie das **Beenden** des Programmes.

Der Menüpunkt **`Bearbeiten`** erlaubt:

- das **Ausschneiden** eines markierten Textbereiches der aktuell in Bearbeitung befindlichen Datei
- das **Kopieren** eines Textbereiches in die Zwischenablage
- das **Einfügen** eines Textbereiches aus der Zwischenablage
- das **Suchen** bzw. das **Suchen und Ersetzen** bestimmter Zeichengruppen im Text

Über den Menüpunkt **`Verschlüsseln`** kann der Nutzer die Verschlüsselung der aktuellen Datei starten. Dazu wird der „public- key“ genutzt, welcher zuvor in der im unteren Bildschirmbereich befindlichen Listbox ausgewählt wurde. Außerdem wird der verschlüsselten Nachricht eine digitale Signatur vorangestellt, mit deren Hilfe der Empfänger Authentizität und Unversehrtheit der Nachricht überprüfen kann.

Über den Menüpunkt **`Entschlüsseln`** kann der Nutzer die Entschlüsselung der aktuellen Datei vornehmen. Dafür wird der private Schlüssel des Benutzers, der sich standardmäßig in der Datei `fc_priv.ini` befindet, verwendet. Bei der Entschlüsselung wird automatisch die digitale Signatur der Nachricht geprüft, bei einer Veränderung der Nachricht wird der Nutzer über diesen Umstand informiert.

Der Menüpunkt **`Optionen`** erlaubt derzeit die Auswahl der Schlüsseldateien für den privaten und die öffentlichen Schlüssel. So können beispielsweise getrennte öffentliche Schlüsselbibliotheken für Privat- und Geschäftskorrespondenz verwaltet werden.

Hilfestellung zum Programm gibt der Menüpunkt **`Hilfe`**. Er bietet:

- allgemeine Informationen zum Programm
- Hinweise zur Bedienung des Programmes
- sowie einige Bemerkungen zum Programm-Support

4.1.2 Die Hilfsmittelleiste

Zur besseren optischen Erfaßbarkeit sind in dieser Zeile die wichtigsten Programmfunktionen mit einem kleinen Sinnbild versehen. So erreicht der Nutzer bei häufiger Programmanwendung ein relativ großes Maß an Zeitersparnis, da er nicht gezwungen ist, eine Auswahl der Programmpunkte über das Menüsystem vorzunehmen.

4.1.3 Der Textbereich

In diesem Bereich wird der zu verschlüsselnde Originaltext angezeigt. Weiterhin sind die Editorbefehle des **`Bearbeiten`**-Menüs mittels eines Klicks auf die rechte Maustaste verfügbar.

4.1.4 Die Listbox mit vorhandenen „public- keys“

In dieser, im unteren Bildschirmbereich befindlichen, Listbox werden die Korrespondenzpartner, deren „public- keys“ der Programmanwender bereits in seinem Besitz ist, mit ihrem Namen und ihrer e-mail Adresse zur Auswahl angezeigt.
Dem Benutzer wird so die Auswahl des Nachrichtenempfängers ermöglicht. Wenn die Anzahl der vorhandenen öffentlichen Schlüssel den darstellbaren Bildschirmbereich überschreitet, kann der Benutzer mit Hilfe eines Rollbalkens die verbleibenden Schlüssel erreichen.
Die dem Anwender bekannten öffentlichen Schlüssel werden in einer Datei mit dem Namen `fc_keys.ini` gespeichert und beim Programmstart von dort in die Listbox eingelesen.

4.1.5 Die Statuszeile

In der Statuszeile wird dem Anwender eine kleine kontextsensitive Hilfe zum aktuellen Programmpunkt gegeben.

4.2 Struktur des Programmes

Da die Programmierung mit Hilfe der Sprache C++ objektorientiert verläuft, stellt das Programm folgende visuelle Objekte(in C++ Klassen genannt) bereit :
Die Klasse TPopupEditFile:
Diese Klasse wurde von der OWL[4]-Klasse TEditFile abgeleitet und stellt ein Fenster zum Bearbeiten von Dateien dar. Außerdem wurde die interne Windows- Antwortfunktion, die auf das Drücken der rechten Maustaste reagiert, so überschrieben, daß sich bei einem solchem Ereignis ein Popup-Menü öffnet, in welchem dem Anwender verschiedene Menübefehle zur Verfügung gestellt werden, z.B. Speichern, Zeichenkette suchen und ersetzen u.s.w.

[4] OWL = „Object Windows Library“ - eine von der Firma Borland entwickelte Klassenbibliothek, in der wichtige Windowsobjekte bereits vordefiniert sind.

Die Klasse TClientLayout:
Diese Klasse wurde von der OWL-Klasse TLayoutWindow abgeleitet und verwaltet den sogenannten Client-Bereich, also den Text- Eingabe- Bereich und die Listbox mit den öffentlichen Schlüsseln.
Diese Listbox befindet sich in einem statischen (nicht verschiebbaren) Fenster.

Die Klasse TFastCryptApp:
Diese Klasse erzeugt das Hauptfenster und stellt damit die eigentliche Windows™- Anwendung dar.
Sie beinhaltet eine Hilfsmittelleiste, die Zugriff auf Sinnbilder (Gadgets) ermöglicht,
eine Statuszeile, die im unteren Bildschirmbereich eine kleine kontextsensitive Hilfe anzeigt und
den Clientbereich, der - wie oben schon beschrieben - den Text- Eingabe- Bereich und die Listbox mit den öffentlichen Schlüsseln einschließt.
Diese Klasse ist unter anderem auch für den Aufruf der Ver- und Entschlüsselungsroutinen verantwortlich.

5 Implementierung und Optimierung der Berechnungen

Alle Berechnungen werden intern mit Dualzahlen durchgeführt, sie sind also auf die Computer-architektur zugeschnitten.
Zum Ausführen der Berechnungen haben wir die Klasse DualZahl implementiert. Sie besteht aus einem Feld, in dem die Dualzahl in word-Blöcken gespeichert wird.
Dieses Feld kann mit unterschiedlichen überlagerten Operatoren (z.B. Addition, Multiplikation...) manipuliert werden.

Zur Veranschaulichung folgt nun beispielhaft die Implementierung der wichtigsten Felder (Eigenschaften) und Funktionen (Methoden) der Klasse DualZahl:

```
typedef unsigned int word;                  // Word: 0..65535 (kein Vorzeichen).
class DualZahl
{
        word Block[AnzBlock];               //  Dieses Feld speichert die Dualzahl.
        int operator<  (DualZahl & op2);    //  Ist-Kleiner-Als-Vergleichs-Operator
        int operator<= (DualZahl & op2);    //  Ist-Kleiner-Gleich-Als-Vergleichs-Operator
        void ShiftLeft(word b);             //  Schiebt die Zahl um b Blöcke nach links.
        void operator<< (word bits);        //  Schiebt die Zahl um <bits> Bits nach links.
        void DZMalWord(DualZahl & a, word x); //  Multipliziert eine DualZahl mit einem word.
        DualZahl operator+ (DualZahl & op2); //  Addition zweier Dualzahlen.
        DualZahl operator- (DualZahl & op2); //  Subtraktion zweier Dualzahlen.
        DualZahl operator* (DualZahl & op2); //  Multiplikation zweier Dualzahlen.
        DualZahl operator% (DualZahl & op2); //  Berechnet Kongruenz zweier Dualzahlen.
       // sowie noch ein paar private Hilfs-Funktionen
```

};

Hinweis:

Die Variable AnzBlock gibt die Anzahl der zum Speichern der Dualzahl verwendeten Words an.
Sie ist im vorliegenden Programm auf 42 gesetzt (= Rechnen mit 200- stelligen Dezimalzahlen).

Die Arbeitsweise der Operatoren und die bei der Implementierung vorgesehenen Optimierungen sollen in diesem Abschnitt genauer unter die Lupe genommen werden.
Um das Kernstück der Ver- und auch der Entschlüsselung, die modulare Exponentation, durchzuführen, benötigt man Algorithmen für die Multiplikation zweier Dualzahlen und für die Berechnung x mod y für zwei Dualzahlen x und y. Die Multiplikation selbst wird auf die Multiplikation einer Dualzahl mit einer Zahl des Datentyps word und der Addition von Dualzahlen zurückgeführt. Die Berechnung der Kongruenz x mod y wird durch fortgesetztes binäres Nach-Links-Schieben (Left-Shift) von y und anschließende Subtraktion des modifizierten y von x erreicht.
Die hier kurz angesprochenen mathematischen Verfahren sollen im folgenden genauer betrachtet werden.

5.1 Addition und Subtraktion von Dualzahlen

Diese Operationen sind prinzipiell einfach , so daß sie keiner größeren Optimierung bedürfen. Zunächst werden die niedrigsten word's (die Felder DualZahl.Block[0]) addiert bzw. voneinander abgezogen, der jeweilige Übertrag wird gemerkt und beim nächsthöheren Feld (in diesem Fall also Block[1]) aufaddiert. Dieses einfache Verfahren wird mit Hilfe einer Schleife solange fortgesetzt, bis alle „Dualziffern" addiert bzw. subtrahiert wurden. Die einzige Optimierung, die hierbei angebracht schien, war die Benutzung des internen Assemblers von Turbo C++, so daß die Rechenoperationen direkt mit den Registern des Prozessors durchgeführt werden.

5.2 Multiplikation einer Dualzahl mit einem Word (Funktion DZMalWord)

Soll die Dualzahl a mit dem Word w multipliziert werden, so benutzen wir folgenden Algorithmus
(Das Produkt von a und w soll mit E bezeichnet werden):
Zunächst wird das niedrigste Word der Dualzahl a (a.Block[0]) mit w multipliziert. Als Ergebnis dieser Multiplikation von zwei Word-Werten erhält man ein Double-Word. Das niederwertige Word dieses Double-Words stellt dann das niedrigste Word des Ergebnisses E dar. Das höherwertige Word wird als Übertrag gespeichert. Im nächsten Schritt wird das nächsthöhere Word der Dualzahl (also a.Block[1]) ebenfalls mit dem Faktor w multipliziert, das Ergebnis dieser Word-Multiplikation ist wieder ein Doppelwort. Nun wird der Übertrag der letzten Multiplikation auf das Doppelwort addiert. Das niederwertige Word des Doppelwords stellt nun den nächsthöheren Block des Ergebnisses E dar. Dieses Verfahren wird in einer Schleife fortgeführt, bis alle „Blöcke" der Dualzahl A multipliziert wurden. Auch hier bot sich eine Verwendung des internen Assemblers von C++ an.

5.3 Multiplikation zweier Dualzahlen

Sollen die beiden Dualzahlen A und B multipliziert werden und wird das Ergebnis mit C bezeichnet, so benutzen wir folgenden Algorithmus: Zunächst wird die Ergebnisvariable C auf Null gesetzt. Es wird eine Hilfs-Dualzahl D benutzt, die Zwischenergebnisse speichert.
In einer Schleife werden für i = 0, 1, 2, ..., AnzBlock - 1 folgende Schritte durchgeführt:

1. D = B * A.Block[i]; - Multipliziere mit Hilfe von DZMalWord() die Dualzahl B mit dem i.ten Word der Dualzahl A.
2. D.ShiftLeft(i); - Schiebe die Hilfs-Dualzahl um i Blöcke nach links; diese Operation entspricht quasi einer Multiplikation von D mit 2 hoch (16 * i).
3. C = C + D; - Addiere auf die Ergebnisvariable C die Hilfsgröße D.

Nach der Schleife steht in C das Ergebnis der Multiplikation von A mit B.

5.4 Berechnung der Kongruenz (A % B)

Stellt die Dualzahl C den Rest der ganzzahligen Division von A durch B dar, so schreibt man:
C = A mod B oder C = A % B.
Zur Bestimmung von C aus Kenntnis der Dualzahlen A und B haben wir unter Ausnutzung eines Algorithmus aus [5] einen schnellen Algorithmus implementiert, der allerdings größere programmiertechnische Kenntnisse voraussetzt. Deshalb haben wir uns entschieden, ihn an einem konkreten Beispiel zu erläutern, damit die prinzipielle Funktionsweise klar wird.

Es sei A = 226, also binär A = 11100010.
Es sei B = 18, also binär B = 00010010.

1. Ist A < B, dann ist C = A. Die Berechnung ist in diesem Fall abgeschlossen.
2. Die Dualzahl B wird solange binär um eine Stelle nach links geschoben, solange A >= B ist. Im Beispiel wird B also um 3 Binärstellen nach links geschoben:
   A = 11100010
   B = 10010000
3. Nun wird B von A abgezogen: A = A - B. Im Beispiel erhält man also:
   A = 11100010 - 10010000 = 01010010
   B = 10010000
4. Nun wird B wieder um eine Stelle nach binär rechts zurückgeschoben: solange bis A >= B oder B den ursprünglichen Wert, den B am Anfang der Berechnung besaß, angenommen hat.
   Im Beispiel gilt nach Schritt (4) also:
   A = 01010010
   B = 01001000

Nun wird wieder zum Schritt (3) gesprungen, d.h. von A wird wieder B abgezogen. Danach wird wieder Schritt (4) ausgeführt u.s.w. Der Algorithmus stoppt, wenn B soweit binär nach rechts geschoben wurde, daß B wieder seinen ursprünglichen Wert angenommen hat. Nun steht in A das Ergebnis C = A % B, weil vom ursprünglichen A immer nur Vielfache von B abgezogen wurden, bis A < B wurde. Im konkreten Beispiel erhält man C = 226 % 18 = 10, also C = 1010 binär.
(Dieses Ergebnis ist richtig, da 226 = 18 * 12 + 10 ist.)

5.5 Modulare Exponentation

Weil der RSA- Algorithmus wie oben beschrieben sehr rechenaufwendig ist, haben wir in unserem Programm Optimierungsmethoden genutzt, um die Rechenzeit zu verkleinern. Ein erster Ansatzpunkt für Optimierungen ist die modulare Exponentation, die Berechnung M^e (MOD n).
Wenn man bei der Berechnung der Potenz M e mal mit sich selbst multiplizieren würde, würde sich bei einer Begrenzung von M und e auf 512 Bits eine Zahl mit bis zu 2^{512} Binärstellen ergeben. Eine Komplettberechnung der Potenz erscheint also wenig sinnvoll. Man kann eine solche große Zahl aber vermeiden, wenn man jeder Einzelmultiplikation eine Modulo- Berechnung folgen läßt [5]. Nun ergeben sich höchstens Zahlen mit einer Länge der doppelten Stellenzahl von n (es gilt immer M < n und e < n).
Um die Rechenzeit zu verkürzen, wird nun folgender Algorithmus zur modularen Exponentation angewendet, für diesen Algorithmus muß e in binärer Form vorliegen.

Man definiert Z als Anzahl der Binärstellen von e. Verwendet wird eine Hilfsvariable C.
Es sei $e = e_{z-1}, e_{z-2}, ..., e_1, e_0$ die Binärzahldarstellung von e mit $0 \leq e_i \leq 1$ und i von 0 bis Z-1.
Die folgenden Schritte werden für i = Z-1, Z-2, ..., 1, 0 durchgeführt:

1. Man berechnet $C = C^2$ MOD n
2. Falls $e_i = 1$ ist, berechnet man C = (C*M) MOD n

Mit diesem Algorithmus reduziert sich die Anzahl der modularen Multiplikationen im Mittel auf 3*Z/2, bei 512-bit Zahlen ergeben sich so im Mittel 768 modulare Multiplikationen.

[5] dies ist möglich, da gilt : (A*B) MOD N = ((A MOD N) * (B MOD N)) MOD N

6 Anhang

Alle im Programm oder der Dokumentation genannten Warenzeichen sind Eigentum ihrer Besitzer und wurden in Programm und Dokumentation ohne Rücksicht auf Schutz- und Patentlage verwendet.

6.1 Eindeutigkeit der Ver- und Entschlüsselung

Nachfolgend soll gezeigt werden, daß die Algorithmen zur Ver- und Entschlüsselung des RSA-Algorithmus bei bekannten öffentlichen *(e,n)* und privatem Schlüssel *(d)* auch wie gewünscht funktionieren. Unsere Nachricht *M* sei eine ganze Zahl für die gelte:

$$0 \pounds M \pounds n-1$$

Nach Definition gilt für Ver- und Entschlüsselungsalgorithmus :

- $\boldsymbol{e}(M)= C= M^e \ (MOD\ n)$
- $\boldsymbol{d}(C)= D= C^d \ (MOD\ n)$

Da $C= M^e \ (MOD\ n)$ gilt auch $C= M^e - sn$, wobei hier *s* eine bestimmte nicht negative ganze Zahl ist. Wenn wir dies in die Definition des Verschlüsselungsalgorithmus einsetzen, gilt:

$$D= C^d \ (MOD\ n)= (M^e - sn)^d \ (MOD\ n)$$

Nach dem Binomialsatz umgeformt erhalten wir :

$$D = M^{ed} \ (MOD\ n).$$

Wenn es uns nun gelänge zu beweisen, daß M= D bzw.

$$M= M^{ed} \ (MOD\ n),$$

so wäre der Beweis für das richtige Funktionieren der Ver- und Entschlüsselungsalgorithmen des RSA- Verfahrens erbracht, da der entschlüsselte Text der Nachricht entspräche. Im folgenden möchten wir versuchen, diesen Beweis zu erbringen:

Da nach Definition gilt :

$$ed= 1 \ (MOD\ (p-1)(q-1))$$

und da $F(n) = (p-1)\ (q-1),$ gilt weiterhin:

$$ed \ MOD \ F(n) = 1$$

Dies läßt sich auch darstellen als:

$$ed = 1+ kF(n) = 1+ k(p-1)\ (q-1)$$

Dabei ist k eine eindeutig bestimmte positive ganze Zahl.

An dieser Stelle muß unterschieden werden, ob m und p teilerfremd sind. Wenn dies zutrifft, kann der Satz von Euler für den Fall $k= p$ (genannt „kleiner Satz von Fermat“) angewendet werden :

Es gilt nach dem Satz von Euler:

$$m^{F(p)} \ MOD\ p= 1$$

Da $F(p) = p-1$ ergibt sich folgender Zusammenhang :

$$M^{ed} \text{ MOD } p = m^{1+kF(n)} \text{ MOD } p = mm^{kF(n)} \text{ MOD } p = mm^{k(p-1)(q-1)} \text{ MOD } p$$
$$= m(m^{p-1})^{k(q-1)} \text{ MOD } p = m1^{k(q-1)} \text{ MOD } p = m \text{ MOD } p$$

Es gilt also:

$$(m^{ed} - m) \text{ MOD } p = 0$$

Nun muß noch der Fall betrachtet werden, in dem *m* und *p* nicht teilerfremd sind:

Da p eine Primzahl ist und somit ein Teiler von m sein muß, gilt :

$$m \text{ MOD } p = 0$$

Wenn p ein Teiler von m ist, so ist p natürlich auch ein Teiler von m^{ed}. Also gilt auch:

$$med \text{ MOD } p = 0$$

Daraus folgt ebenfalls:

$$(m^{ed} - m) \text{ MOD } p = 0$$

Es ist also erkennbar, daß obige Aussage für den Fall daß p und m teilerfremd sind ebenso gilt, wie für den Fall, daß p und m nicht teilerfremd sind.

Da die Herleitung dieser Aussage für alle Primzahlen p angelegt war, läßt sich auch sagen:

$$(m^{ed} - m) \text{ MOD } q = 0$$

Nach den obigen Schritten gilt :

p teilt $(m^{ed} - m)$ und *q* teilt $(m^{ed} - m)$. Wenn die beiden verschiedenen Primzahlen *p* und *q* Teiler von $(m^{ed} - m)$ sind, so muß auch ihr Produkt ein Teiler dieser Zahl sein.
Dies führt uns zu der Aussage:

$$m^{ed} \text{ MOD } pq = m$$

und da n= pq gilt somit

$$\boldsymbol{m = m^{ed} \text{ MOD } n}$$

q.e.d.

Damit ist bewiesen, daß die Ver- und Entschlüsselungsverfahren des RSA- Algorithmus in jedem Fall fehlerfrei funktionieren, da die Entschlüsselung der verschlüsselten Nachricht offensichtlich wieder zur Originalnachricht führt.

6.2 Literatur

1. Beutelspacher, Albrecht; „Kryptologie“ ;Wiesbaden 1993
2. Buchmann, Johannes; „Faktorisierung großer Zahlen“ ; in: „Spektrum der Wissenschaft“ ; Heft 9/96 Seiten 80-88; September 1996
3. Hagemann, Hagen; Rieke, Andreas; „Datenschlösser“ ; in: „c´t“ ; Heft 8/94 Seiten 230-283; 1994
4. Diffie, Whitfield; Hellman, Martin E.; „New Directions in Cryptography“ , in : „IEE Transactions on Information Theory“ ; VOL. IT-22 No. 6; November 1976
5. Engel, Arthur, „Mathematisches Experimentieren mit dem PC“ ; Stuttgart 1991
6. Rivest, Ronald L.; Shamir, Adi; Adleman, Leonard; „A Method for Obtaining Digital Signatures and Public Key Cryptosystems“ ; in: „Communications of the ACM“ ; Volume 21, Number 2; February 1978
7. Rivest, Ronald L.; Shamir, Adi; Adleman, Leonard; „Cryptographic communications system and method“ ; US- Patentschrift 4,405,829; 1983
8. Sedgewick, Robert; „Algorithmen“ ; Reading 1995
9. Schönleber, Claus; „Verschlüsselungsverfahren für PC- Daten“ ; Poing 1995
10. Seidler, Christoph; „Kryptosysteme mit öffentlichem Schlüssel“ ; Halle (Saale) 1996

6.3 Danksagungen

Wir möchten uns hiermit herzlich bei Dr. A. Koch, Georg- Cantor- Gymnasium Halle, für die uns erwiesene Unterstützung zur Literaturbeschaffung bedanken. Ebenso bedanken wir uns bei Ch. Winger, Georg- Cantor- Gymnasium Halle, für die Betreuung im Rahmen des Wettbewerbes „Jugend forscht“ . Bedanken möchten wir uns ebenfalls bei Dr. Hintz, Otto v. Guericke- Universität Magdeburg, für die Vorbereitung auf den Bundeswettbewerb „Jugend forscht“ .