Chiffrierung nach ELGAMAL

Christian Koch
96NKT

13. Januar 2000

Zusammenfassung

Im folgenden die Aufgabenstellung zum Projekt 6 „Programmierung des ELGAMAL-Verfahrens“ im Wahlpflichtfach Kryptologie von Prof. Jahn am Fb IMN der HTWK Leipzig.

Vorgegeben seien eine natürliche Zahl L sowie ein Klartext $\mathscr{P}$. Zu erzeugen sind zunächst eine Primzahl p mit L Bits, so daß auch $(p-1)/2$ eine Primzahl ist, sowie ein primitives Element $\alpha \in \mathbf{Z}_p^*$. Weiterhin ist ein a mit $2 \leq a \leq p-2$ als geheimer Schlüssel auszuwählen.

Der Klartext $\mathscr{P}$ ist in Blöcke der Länge $L-1$ Bit zu zerlegen und nach dem ELGAMAL-Verfahren zu verschlüsseln!

Sowohl Ver- als auch Entschlüsselung sind zu programmieren!

1 Vorbetrachtungen

Das ELGAMAL-Verfahren ist ein asymmetrisches Chiffrierverfahren, d. h., zur Entschlüsselung ist ein anderer Schlüssel notwendig als zur Verschlüsselung. Derartige Chiffrierverfahren werden aber aufgrund ihrer Rechenintensität nur in hybriden Systemen eingesetzt, so daß damit nur ein symmetrischer Sitzungsschlüssel oder ein MAC[1] geschützt wird.

Das Schlüsselpaar wird folgendermaßen erzeugt:

1. Bestimme eine große Primzahl p.
2. Bestimme ein primitives Element α, auch Generator[2] genannt, bzgl. $\mathbf{Z}_p^*$.
3. Wähle ein beliebiges a mit $1 \leq a \leq p-2$.
4. Berechne $\beta := \alpha^a \bmod p$.

[1] Message Authentication Code
[2] $\mathbf{Z}_p^* = \{\alpha^i \bmod p \mid 0 \leq i \leq p-2\}$

Der öffentliche Schlüssel ist dann $\mathscr{K}_{\text{pub}} := (p, \alpha, \beta)$ und der private $\mathscr{K}_{\text{prv}} := (p, a)$.

Sei nun $x \in \mathscr{P}$ mit $0 \leq x \leq p-1$ ein beliebiger Klartextblock. Dann wird wie folgt chiffriert:

$$\begin{aligned} \mathscr{E}_{\mathscr{K}_{\text{pub}}}(x) &:= (y_1, y_2) \\ y_1 &:= \alpha^{\lambda} \bmod p \\ y_2 &:= x \cdot \beta^{\lambda} \bmod p \end{aligned}$$

Wie man leicht sieht, ist der Geheimtext doppelt so lang wie der Klartext. Dies ist auch ein Grund dafür, weshalb dieses Verfahren ausschließlich in hybriden Systemen benutzt wird. Das λ ist für jeden Block zufällig zu wählen mit $1 \leq \lambda \leq p-2$.

Den Klartext erhält man wieder durch:

$$\begin{aligned} \mathscr{D}_{\mathscr{K}_{\text{prv}}}(y_1, y_2) &:= y_2 \cdot y_1^{-a} \bmod p \\ &= x \end{aligned}$$

2 Implementierung

Entscheidend für die Programmierung ist die effiziente Verarbeitung sehr großer natürlicher Zahlen. ARIBAS bietet hierfür zwar eine sehr gute Unterstützung, jedoch keine Interaktion mit Hilfe einer grafischen Oberfläche.

Nach einer Suche im Internet fiel die Wahl auf die GNU Multiple Precision Library (`http://www.swox.com/gmp/`) in der aktuellen Version 2.0.2. Sie ist in C geschrieben und auch unter MS Windows einsetzbar. Die zugehörige Dokumentation liegt diesem Projekt als HTML-Datei bei.

Das Programm wurde im wesentlichen auf zwei Dateien aufgeteilt. In `main.c` wird die gesamte Benutzerinteraktion durchgeführt. In `fktn.c` sind alle Funktionen untergebracht, die die Stringumwandlung in Zahlen und die kryptographischen Berechnungen durchführen. Das Benutzerinterface, welches in Abbildung 1 dargestellt ist, wird in der Datei `main.rc` definiert.

Noch etwas zur Bezeichnungskonvention der Variablen. Ein Präfix kennzeichnet den Typ der Variablen, sofern dieser keinen Standard darstellt. Ein Suffix `_p` kennzeichnet die Variable als Pointer.

2.1 makefile

Das `makefile` dient dem automatischen Erzeugen des Programms. Durch Eingabe von `make clean` kann das Quellverzeichnis in den Ausgangszustand zurückversetzt werden.

```
CC      = gcc
LD      = gcc
CFLAGS = -O2 -Wall
LDFLAGS = -mwindows -s
```

```
LIBS = -lcomctl32 -lgmps -liberty
OBJS = main.o fktn.o res.o

default: elgamal.exe

elgamal.exe : $(OBJS) makefile
        $(LD) $(LDFLAGS) -o $@ $(OBJS) $(LIBS)

.c.o:
        $(CC) $(CFLAGS) -c $<

res.o: main.res makefile
        windres $< $@

main.res: main.rc makefile
        rc $<

clean:
        del *.exe
        del *.o
        del *.res
```

2.2 global.h

Die Datei `global.h` definiert die Typen für das Schlüsselpaar.

```
typedef struct
{
    mpz_t p;
    mpz_t alpha;
    mpz_t beta;
} PubKey;


typedef struct
{
    mpz_t p;
    mpz_t a;
} SecKey;
```

2.3 id.h

In dieser Datei werden die Zahlenkonstanten für die Fensterelemente definiert.

```
/* -- Push Buttons -- */
#define IDB_Chiffrieren     100
#define IDB_Dechiffrieren   101
#define IDB_neu             102
#define IDB_SPK             103
#define IDB_SGT             104
```

```
/* ---- Edit Text ---- */
#define IDE_Klartext        200
#define IDE_Geheimtext      201
#define IDE_alpha           202
#define IDE_beta            203
#define IDE_p               204
#define IDE_a               205
#define IDE_Bitlaenge       206
```

2.4 fktn.h

Diese Headerdatei ist für die Deklaration der Funktionsprototypen in `fktn.c` verantwortlich.

```
void
genkeypair(PubKey *, SecKey *, int const);

mpz_t *
encrypt(mpz_t * const, PubKey const, int const);

mpz_t *
decrypt(mpz_t * const, SecKey const, int const);

void
nextstrongprime(mpz_t, mpz_t const);

mpz_t *
str2arr(int * const, unsigned char const *, int const);

unsigned char *
arr2str(int const, mpz_t * const, int const);
```

2.5 fktn.c

Diese Quelldatei definiert Funktionen, die alle Berechnungen vornimmt, die nichts mit dem Benutzerinterface zu tun haben.

genkeypair erzeugt ein Schlüsselpaar, wobei der Modul `p` eine vorgebene Länge `L` und die Eigenschaft einer sog. starken Primzahl besitzt.

encrypt verschlüsselt `no_of_blocks` Klartextblöcke `mpzIn_p` unter Verwendung des öffentlichen Schlüssels `pk`.

decrypt entschlüsselt `no_of_blocks` Geheimtextblöcke `mpzIn_p` unter Verwendung des privaten Schlüssels `sk`.

nextstrongprime findet die nächste starke Primzahl `mpzPrime`, die auf `mpzNumber` folgt.

str2arr wandelt eine Zeichenkette `str_p` in `no_of_blocks` Zahlenblöcke der Länge `bit_per_block` um.

arr2str wandelt `no_of_blocks` Zahlenblöcke `mpzarr_p` der Länge `bit_per_block` in eine Zeichenkette um.

```
#include <windows.h>
#include <math.h>
#include <gmp.h>
#include "global.h"
#include "fktn.h"


void
genkeypair(PubKey *pk, SecKey *sk, int const L)
{
    mpz_t mpzMaske;

    /* Einser-Maske der Länge L erzeugen */
    mpz_init_set_ui(mpzMaske, 1);
    mpz_mul_2exp(mpzMaske, mpzMaske, L);
    mpz_sub_ui(mpzMaske, mpzMaske, 1);

    /* Bestimmung einer starken Primzahl p der Länge L Bit */
    srandom(time(NULL));
    do
    {
        mpz_random(pk->p, mpz_size(mpzMaske) + 2);
        mpz_and(pk->p, pk->p, mpzMaske);
        nextstrongprime(pk->p, pk->p);
    }
    while (mpz_sizeinbase(pk->p, 2) != L);
    mpz_set(sk->p, pk->p);

    /* Wahl eines primitiven Elementes alpha bzgl. Zp */
    srandom(time(NULL));
    do
    {
        do
        {
            mpz_random(pk->alpha, mpz_size(pk->p));
            mpz_and(pk->alpha, pk->alpha, mpzMaske);
        }
        while (mpz_cmp(pk->alpha, pk->p) > 0);

        /* p - 1 = [2] * [(p - 1) /2] */
        mpz_powm_ui(pk->beta, pk->alpha, 2, pk->p);
        if (mpz_cmp_ui(pk->beta, 1) == 0)
        {
            continue;
        }
        mpz_sub_ui(pk->beta, pk->p, 1);
        mpz_tdiv_q_2exp(pk->beta, pk->beta, 1);
```

```
        mpz_powm(pk->beta, pk->alpha, pk->beta, pk->p);
    }
    while (mpz_cmp_ui(pk->beta, 1) == 0);

    /* Bestimmung eines a mit 1 <= a <= p-2 */
    mpz_sub_ui(pk->beta, pk->p, 2);
    do
    {
        mpz_random(sk->a, mpz_size(pk->p));
        mpz_and(sk->a, sk->a, mpzMaske);
    }
    while (mpz_cmp(sk->a, pk->beta) > 0);
    mpz_clear(mpzMaske);

    /* Bestimmung des beta gemäß beta = alpha ** a mod p */
    mpz_powm(pk->beta, pk->alpha, sk->a, pk->p);

    return;
}


mpz_t *
encrypt(mpz_t * const mpzIn_p, PubKey const pk, int const no_of_blocks)
{
    int i;
    mpz_t mpztemp, mpzMaske, mpzlambda, *mpzOut_p;

    /* Anforderung von Speicher und Initialisierung der Variablen */
    mpzOut_p = (mpz_t *) VirtualAlloc(NULL, 2 * no_of_blocks * sizeof (mpz_t),\
                                      MEM_COMMIT, PAGE_READWRITE);
    if (mpzOut_p == NULL)
    {
        return (mpzOut_p);
    }

    for (i = 0; i < 2 * no_of_blocks; i++)
    {
        mpz_init(mpzOut_p[i]);
    }

    /* Einser-Maske der Länge L erzeugen */
    mpz_init_set_ui(mpzMaske, 1);
    mpz_mul_2exp(mpzMaske, mpzMaske, mpz_sizeinbase(pk.p, 2));
    mpz_sub_ui(mpzMaske, mpzMaske, 1);

    /* Verschlüsselung der einzelnen Blöcke */
    mpz_init(mpztemp);
    mpz_init(mpzlambda);
    srandom(time(NULL));
    for (i = 0; i < no_of_blocks; i++)
    {
        /* Bestimmung eines lambda mit 1 <= lambda <= p-2 */
        mpz_sub_ui(mpztemp, pk.p, 2);
```

```
        do
        {
            mpz_random(mpzlambda, mpz_size(pk.p));
            mpz_and(mpzlambda, mpzlambda, mpzMaske);
        }
        while (mpz_cmp(mpzlambda, mpztemp) > 0);

        /* Berechnung nach ElGamal */
        mpz_powm(mpzOut_p[2*i], pk.alpha, mpzlambda, pk.p);
        mpz_powm(mpzOut_p[2*i+1], pk.beta, mpzlambda, pk.p);
        mpz_mul(mpzOut_p[2*i+1], mpzOut_p[2*i+1], mpzIn_p[i]);
        mpz_mod(mpzOut_p[2*i+1], mpzOut_p[2*i+1], pk.p);
    }
    mpz_clear(mpztemp);
    mpz_clear(mpzMaske);
    mpz_clear(mpzlambda);

    return mpzOut_p;
}


mpz_t *
decrypt(mpz_t * const mpzIn_p, SecKey const sk, int const no_of_blocks)
{
    int i;
    mpz_t *mpzOut_p;

    /* Anforderung von Speicher und Initialisierung der Variablen */
    mpzOut_p = (mpz_t *) VirtualAlloc(NULL, no_of_blocks * sizeof (mpz_t),\
                                      MEM_COMMIT, PAGE_READWRITE);
    if (mpzOut_p == NULL)
    {
        return (mpzOut_p);
    }

    for (i = 0; i < no_of_blocks; i++)
    {
        mpz_init(mpzOut_p[i]);
    }

    /* Entschlüsselung der einzelnen Blöcke */
    for (i = 0; i < no_of_blocks; i++)
    {
        /* Berechnung nach ElGamal */
        mpz_powm(mpzOut_p[i], mpzIn_p[2*i], sk.a, sk.p);
        mpz_invert(mpzOut_p[i], mpzOut_p[i], sk.p);
        mpz_mul(mpzOut_p[i], mpzOut_p[i], mpzIn_p[2*i+1]);
        mpz_mod(mpzOut_p[i], mpzOut_p[i], sk.p);
    }

    return mpzOut_p;
}
```

```
void
nextstrongprime(mpz_t mpzPrime, mpz_t const mpzNumber)
{                                                                    160
    mpz_t mpztemp;

    /* ist mpzNumber gerade? */
    mpz_set(mpzPrime, mpzNumber);
    if ((mpz_get_ui(mpzPrime) % 2) == 0)
    {
        mpz_add_ui(mpzPrime, mpzPrime, 1);
        if (mpz_probab_prime_p(mpzPrime, 25) == 1)
        {
            return;                                                  170
        }
    }

    /* Suche nach einer Primzahl p, so daß auch (p-1)/2 prim ist */
    mpz_init(mpztemp);
    do
    {
        mpz_add_ui(mpzPrime, mpzPrime, 2);
        while (mpz_probab_prime_p(mpzPrime, 25) != 1)
        {                                                            180
            mpz_add_ui(mpzPrime, mpzPrime, 2);
        }
        mpz_sub_ui(mpztemp, mpzPrime, 1);
        mpz_tdiv_q_2exp(mpztemp, mpztemp, 1);
    }
    while (mpz_probab_prime_p(mpztemp, 25) != 1);
    mpz_clear(mpztemp);

    return;
}                                                                    190


mpz_t *
str2arr(int * const no_of_blocks_p, unsigned char const *str_p, int const bits_per_block)
{
    int i, bits_left, str_len;
    mpz_t mpztemp, *mpzarr_p;

    /* Bestimmung der benötigten Anzahl Blöcke */                    200
    str_len = lstrlen(str_p);
    *no_of_blocks_p = ceil((double) (str_len * 8) / bits_per_block);

    /* Anforderung von Speicher und Initialisierung der Variablen */
    mpzarr_p = (mpz_t *) VirtualAlloc(NULL, *no_of_blocks_p * sizeof (mpz_t),\
                                      MEM_COMMIT, PAGE_READWRITE);
    if (mpzarr_p == NULL)
    {
        return (mpzarr_p);
```

```
    }

    for (i = 0; i < *no_of_blocks_p; i++)
    {
        mpz_init(mpzarr_p[i]);
    }

    /* Aufteilung des Strings in Blöcke der Länge bits_per_block */
    mpz_init(mpztemp);
    for (i = 0; i < str_len; i++)
    {
        bits_left = bits_per_block - (i * 8) % bits_per_block;
        if (bits_left < 8)
        {
            /* Aufteilung des Zeichens auf zwei Blöcke */
            mpz_set_ui(mpztemp, str_p[i] >> (8 - bits_left));
            mpz_add(mpzarr_p[(i * 8) / bits_per_block],\
                    mpzarr_p[(i * 8) / bits_per_block], mpztemp);

            mpz_set_ui(mpztemp, ((str_p[i] << bits_left) & 255) >> bits_left);
            mpz_mul_2exp(mpztemp, mpztemp, bits_per_block - (8 - bits_left));
            mpz_add(mpzarr_p[((i * 8) / bits_per_block) + 1],\
                    mpzarr_p[((i * 8) / bits_per_block) + 1], mpztemp);
        }
        else
        {
            /* Abbildung des Zeichen in einen Block */
            mpz_set_ui(mpztemp, str_p[i]);
            mpz_mul_2exp(mpztemp, mpztemp, bits_left - 8);
            mpz_add(mpzarr_p[(i * 8) / bits_per_block],\
                    mpzarr_p[(i * 8) / bits_per_block], mpztemp);
        }
    }
    mpz_clear(mpztemp);

    return mpzarr_p;
}



unsigned char *
arr2str(int const no_of_blocks, mpz_t * const mpzarr_p, int const bits_per_block)
{
    unsigned char *str_p;
    int i, bits_left, str_len;
    mpz_t mpztemp;

    /* Bestimmung der Anzahl der benötigten Zeichen */
    str_len = (no_of_blocks * bits_per_block) / 8;

    /* Anforderung von Speicher */
    str_p = (unsigned char *) VirtualAlloc(NULL, str_len * sizeof (unsigned char) + 1,\
                                           MEM_COMMIT, PAGE_READWRITE);
    if (str_p == NULL)
```

```
    {
        return (str_p);
    }

    /* Rekonstruktion der Zeichenkette */
    mpz_init(mpztemp);
    for (i = 0; i < str_len; i++)                                          270
    {
        /* Berechnung freier Bits im aktuellen Block */
        bits_left = bits_per_block - (i * 8) % bits_per_block;

        if (bits_left < 8)
        {
            /* Zusammensetzung eines Zeichens aus zwei Blöcken */
            str_p[i] = mpz_get_ui(mpzarr_p[(i * 8) / bits_per_block]);
            str_p[i] <<= (8 - bits_left);
                                                                           280
            mpz_tdiv_q_2exp(mpztemp, mpzarr_p[((i * 8) / bits_per_block) + 1],\
                            bits_per_block - (8 - bits_left));
            str_p[i] += mpz_get_ui(mpztemp);
        }
        else
        {
            /* Abbildung eines Teils des Blockes auf ein Zeichen */
            mpz_tdiv_q_2exp(mpztemp, mpzarr_p[(i * 8) / bits_per_block], bits_left - 8);
            str_p[i] = mpz_get_ui(mpztemp);
        }                                                                  290
    }
    mpz_clear(mpztemp);

    str_p[str_len] = '\0';
    return str_p;
}
```

2.6 main.h

Die Datei `main.h` deklariert die Funktionsprototypen, die in `main.c` benutzt werden.

```
/* maximale Länge des Moduls p */
#define MAX_BIT 200

BOOL CALLBACK
DlgProc (HWND, UINT, WPARAM, LPARAM);

BOOL
Cls_OnInitDialog (HWND, HWND, LPARAM);

void                                                                       10
Cls_OnCommand (HWND, int, HWND, UINT);

void
```

```
Cls_OnSysCommand (HWND, UINT, int, int);
```

2.7 main.rc

In folgender Ressourcen-Definitionsdatei wird vor allem das Benutzerinterface definiert.

```
#include <windows.h>
#include "id.h"

elgamal ICON elgamal.ico

VS_VERSION_INFO VERSIONINFO
FILEVERSION 1,00,00,00
PRODUCTVERSION 1,00,0,0
FILEFLAGS 0 /* VS_FF_PRERELEASE */
FILEOS VOS__WINDOWS32
FILETYPE VFT_APP
{
    BLOCK "StringFileInfo"
    {
        BLOCK "040704e4"
        {
            VALUE "FileDescription",
                 "Asymmetrische Chiffrierung nach ElGamal"
            VALUE "FileVersion", "1.00"
            VALUE "InternalName", "elgamal"
            VALUE "LegalCopyright",
                 "Copyright © 1999/2000 Christian Koch."
            VALUE "OriginalFilename", "elgamal.exe"
            VALUE "ProductName", "ElGamal"
            VALUE "ProductVersion", "1.00"
            VALUE "Kommentar",
                 "Dieses Programm unterliegt der GPL V2 oder höher."
        }
    }
    BLOCK "VarFileInfo"
    {
        VALUE "Translation", 0x0407, 1252
    }
}


ElGamal DIALOG 32768, 0, 384, 277
STYLE WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "ElGamal"
FONT 8, "MS Sans Serif"
{
    GROUPBOX     "Nachricht", -1, 5, 5, 370, 145
    LTEXT        "Klartext", -1, 15, 20, 25, 8
    EDITTEXT     IDE_Klartext, 15, 30, 150, 110, ES_MULTILINE |
                 ES_AUTOHSCROLL | ES_AUTOVSCROLL | ES_WANTRETURN |
```

```
                WS_HSCROLL| WS_VSCROLL
    PUSHBUTTON  "-->", IDB_Chiffrieren, 175, 65, 30, 14
    PUSHBUTTON  "<--", IDB_Dechiffrieren, 175, 90, 30, 14
    LTEXT       "Geheimtext", -1, 215, 20, 40, 8
    EDITTEXT    IDE_Geheimtext, 215, 30, 150, 110, ES_MULTILINE |                50
                ES_AUTOVSCROLL | WS_VSCROLL
    GROUPBOX    "Parameter", -1, 5, 155, 370, 115
    LTEXT       "alpha", -1, 15, 170, 20, 8
    EDITTEXT    IDE_alpha, 15, 180, 110, 35, ES_MULTILINE | ES_AUTOVSCROLL
    LTEXT       "beta", -1, 15, 220, 20, 8
    EDITTEXT    IDE_beta, 15, 230, 110, 30, ES_MULTILINE | ES_AUTOVSCROLL
    LTEXT       "p", -1, 140, 170, 5, 8
    EDITTEXT    IDE_p, 140, 180, 100, 35, ES_MULTILINE | ES_AUTOVSCROLL
    LTEXT       "Bitlänge: ", -1, 140, 231, 30, 8
    EDITTEXT    IDE_Bitlaenge, 180, 230, 60, 12, ES_NUMBER                    60
    DEFPUSHBUTTON "erzeuge neues Schlüsselpaar", IDB_neu, 140, 245, 100, 14
    LTEXT       "a", -1, 255, 170, 5, 8
    EDITTEXT    IDE_a, 255, 180, 110, 35, ES_MULTILINE | ES_AUTOVSCROLL
    PUSHBUTTON  "Public-Key speichern", IDB_SPK, 255, 230, 110, 14
    PUSHBUTTON  "Geheimtext speichern", IDB_SGT, 255, 245, 110, 14
}
```

2.8 main.c

In `main.c` wird die gesamte Benutzerinteraktion behandelt. Somit ist es relativ einfach, das Chiffierverfahren auf ein anderes Betriebssystem zu portieren.

Dem Programm kann ein Befehlszeilenargument übergeben werden, welches die Zahlenbasis für die Darstellung der Schlüsselparameter vorgibt. Wird das Programm z. B. durch `elgamal 2` gestartet, dann werden alle Angaben im Binärsystem dargestellt. Vorgabe ist aber das Dezimalsystem.

Die Schlüsseldaten können vorgegeben werden, oder es kann auch ein neues Schlüsselpaar einer bestimmten Länge des Moduls p generiert werden. Zum gegenwärtigen Zeitpunkt habe ich die Bitlänge aufgrund der Berechnungsdauer auf 200 begrenzt. Die Daten werden unter Benutzung der eingetragenen Schlüsseldaten ver- oder entschlüsselt.

```
#include <windows.h>
#include <windowsx.h>
#include <commdlg.h>
#include <stdlib.h>
#include <gmp.h>
#include "global.h"
#include "main.h"
#include "fktn.h"
#include "id.h"                                                            10

HINSTANCE hActInstance;
char szAppName[] = "ElGamal";
```

```
/* Zahlenbasis für Aus- und Eingabe der Parameter */
int BASE;

/* Schlüsselpaar */
PubKey pk;
SecKey sk;                                                          20



int WINAPI
WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine,\
        int iCmdShow)
{
    hActInstance = hInstance;

    /* auf Befehlszeilenargument überprüfen */                      30
    BASE = atoi(szCmdLine);
    if ((BASE < 2) || (BASE > 36))
    {
        BASE = 10;
    }

    return DialogBox(hInstance, szAppName, NULL, DlgProc);
}

                                                                    40

BOOL CALLBACK
DlgProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        HANDLE_MSG(hwnd, WM_INITDIALOG, Cls_OnInitDialog);
        HANDLE_MSG(hwnd, WM_COMMAND, Cls_OnCommand);
        HANDLE_MSG(hwnd, WM_SYSCOMMAND, Cls_OnSysCommand);
    }                                                               50

    return FALSE;
}



BOOL
Cls_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)
{
    HICON hIcon = LoadIcon(hActInstance, szAppName);                60

    SendMessage(hwnd, WM_SETICON, (WPARAM) ICON_SMALL, (LPARAM) hIcon);

    /* Initialisierung des Schlüsselpaars */
    mpz_init(pk.p);
    mpz_init(pk.alpha);
    mpz_init(pk.beta);
    mpz_init(sk.p);
```

```
    mpz_init(sk.a);

    return FALSE;
}


void
Cls_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
    char szbuf[MAX_BIT + 1], *buf_p;
    int i, j, L, str_len, no_of_blocks;
    mpz_t *mpztemp_p = NULL, *mpztemp2_p;
    OPENFILENAME sfn;
    HANDLE hFile;
    DWORD dwBytes;


    switch (id)
    {
        case IDB_neu:
        /* Bitlänge bestimmen und auf Grenzen überprüfen */
        L = GetDlgItemInt(hwnd, IDE_Bitlaenge, NULL, FALSE);
        if ((L < 20) || (L > MAX_BIT))
        {
            MessageBox(hwnd, "20 <= Bitlänge <= 200", szAppName,\
                    MB_OK | MB_ICONINFORMATION);
            break;
        }
        /* Schlüsselpaar der Länge L Bit erzeugen */
        genkeypair(&pk, &sk, L);

        /* Schlüsseldaten in die Editboxen eintragen */
        mpz_get_str(szbuf, BASE, pk.p);
        SetDlgItemText(hwnd, IDE_p, szbuf);

        mpz_get_str(szbuf, BASE, pk.alpha);
        SetDlgItemText(hwnd, IDE_alpha, szbuf);

        mpz_get_str(szbuf, BASE, sk.a);
        SetDlgItemText(hwnd, IDE_a, szbuf);

        mpz_get_str(szbuf, BASE, pk.beta);
        SetDlgItemText(hwnd, IDE_beta, szbuf);
        break;


        case IDB_Chiffrieren:
        /* alpha aus Editbox auslesen */
        str_len = SendDlgItemMessage(hwnd, IDE_alpha, WM_GETTEXTLENGTH, 0, 0);
        if (str_len == 0)
        {
            MessageBox(hwnd, "Bitte alpha eingeben.", szAppName,\
                    MB_OK | MB_ICONINFORMATION);
```

```
    break;
}
GetDlgItemText(hwnd, IDE_alpha, szbuf, sizeof (szbuf));
mpz_set_str(pk.alpha, szbuf, BASE);

/* beta aus Editbox auslesen */
str_len = SendDlgItemMessage(hwnd, IDE_beta, WM_GETTEXTLENGTH, 0, 0);
if (str_len == 0)                                                    130
{
    MessageBox(hwnd, "Bitte beta eingeben.", szAppName,\
            MB_OK | MB_ICONINFORMATION);
    break;
}
GetDlgItemText(hwnd, IDE_beta, szbuf, sizeof (szbuf));
mpz_set_str(pk.beta, szbuf, BASE);

/* p aus Editbox auslesen */
str_len = SendDlgItemMessage(hwnd, IDE_p, WM_GETTEXTLENGTH, 0, 0);   140
if (str_len == 0)
{
    MessageBox(hwnd, "Bitte p eingeben.", szAppName,\
            MB_OK | MB_ICONINFORMATION);
    break;
}
GetDlgItemText(hwnd, IDE_p, szbuf, sizeof (szbuf));
mpz_set_str(pk.p, szbuf, BASE);

/* L aus p bestimmen und in Editbox eintragen */                     150
L = mpz_sizeinbase(pk.p, 2);
SetDlgItemInt(hwnd, IDE_Bitlaenge, L, FALSE);

/* Bestimmung der Länge des Klartextstrings */
str_len = SendDlgItemMessage(hwnd, IDE_Klartext, WM_GETTEXTLENGTH, 0, 0);
if (str_len == 0)
{
    MessageBox(hwnd, "Bitte Klartext eingeben.", szAppName,\
            MB_OK | MB_ICONINFORMATION);
    break;                                                           160
}

/* Anfordern von Speicher für den Klartextstring */
buf_p = (char *) VirtualAlloc(NULL, ++str_len * sizeof (char),\
                              MEM_COMMIT, PAGE_READWRITE);

/* Klartext aus Editbox auslesen */
GetDlgItemText(hwnd, IDE_Klartext, buf_p, str_len);

/* Klartext in Blöcke der Länge L-1 aufteilen */                     170
mpztemp_p = str2arr(&no_of_blocks, buf_p, L-1);
VirtualFree(buf_p, 0, MEM_RELEASE);

/* Klartextblöcke mit pk verschlüsseln */
mpztemp2_p = encrypt(mpztemp_p, pk, no_of_blocks);
```

```
/* Freigeben des Speichers der Klartextblöcke */
for (i = 0; i < no_of_blocks; i++)
{
    mpz_clear(mpztemp_p[i]);                                          180
}
VirtualFree(mpztemp_p, 0, MEM_RELEASE);

/* Bestimmung der Länge des Geheimtextstrings */
no_of_blocks *= 2;
str_len = 0;
for (i = 0; i < no_of_blocks; i++)
{
    str_len += mpz_sizeinbase(mpztemp2_p[i], BASE);
}                                                                     190

/* Anfordern von Speicher für den Geheimtextstring */
buf_p = (char *) VirtualAlloc(NULL, (++str_len + no_of_blocks) *\
                                    sizeof (char),\
                                    MEM_COMMIT, PAGE_READWRITE);
/* buf_p[0] = '\0'; VirtualAlloc initialisiert Speicher mit 0 */

/* Umwandlung der Geheimtextblöcke in den Geheimtextstring */
for (i = 0; i < no_of_blocks; i++)
{                                                                     200
    mpz_get_str(szbuf, BASE, mpztemp2_p[i]);
    lstrcat(buf_p, szbuf);
    lstrcat(buf_p, "_");
}

/* Freigeben des Speichers der Geheimtextblöcke */
for (i = 0; i < no_of_blocks; i++)
{
    mpz_clear(mpztemp2_p[i]);
}                                                                     210
VirtualFree(mpztemp2_p, 0, MEM_RELEASE);

/* Ausgabe der Geheimtextstrings */
SetDlgItemText(hwnd, IDE_Geheimtext, buf_p);

/* Freigabe des Speichers des Geheimtextstrings */
VirtualFree(buf_p, 0, MEM_RELEASE);
break;

                                                                      220
case IDB_Dechiffrieren:
/* p aus Editbox auslesen */
str_len = SendDlgItemMessage(hwnd, IDE_p, WM_GETTEXTLENGTH, 0, 0);
if (str_len == 0)
{
    MessageBox(hwnd, "Bitte p eingeben.", szAppName,\
               MB_OK | MB_ICONINFORMATION);
    break;
}
GetDlgItemText(hwnd, IDE_p, szbuf, ++str_len);                        230
```

```
mpz_set_str(sk.p, szbuf, BASE);

/* a aus Editbox auslesen */
str_len = SendDlgItemMessage(hwnd, IDE_a, WM_GETTEXTLENGTH, 0, 0);
if (str_len == 0)
{
    MessageBox(hwnd, "Bitte a eingeben.", szAppName,\
               MB_OK | MB_ICONINFORMATION);
    break;
}                                                                    240
GetDlgItemText(hwnd, IDE_a, szbuf, ++str_len);
mpz_set_str(sk.a, szbuf, BASE);

/* L aus p bestimmen und in Editbox eintragen */
L = mpz_sizeinbase(sk.p, 2);
SetDlgItemInt(hwnd, IDE_Bitlaenge, L, FALSE);
if ((L < 20) || (L > MAX_BIT))
{
     MessageBox(hwnd, "20 <= Bitlänge <= 200", szAppName,\
                MB_OK | MB_ICONINFORMATION);                         250
     break;
}

/* Bestimmung der Länge des Geheimtextes */
str_len = SendDlgItemMessage(hwnd, IDE_Geheimtext, WM_GETTEXTLENGTH, 0, 0);
if (str_len == 0)
{
    MessageBox(hwnd, "Bitte Geheimtext eingeben.", szAppName,\
               MB_OK | MB_ICONINFORMATION);
    break;                                                           260
}

/* Anfordern von Speicher für den Geheimtextstring */
buf_p = (char *) VirtualAlloc(NULL, ++str_len * sizeof (char),\
                              MEM_COMMIT, PAGE_READWRITE);

/* Geheimtext aus Editbox auslesen */
GetDlgItemText(hwnd, IDE_Geheimtext, buf_p, str_len);

/* Bestimmung der Anzahl der Geheimtextblöcke */                     270
no_of_blocks = 0;
i = 0;
while (buf_p[i] != '\0')
{
    if (buf_p[i] == '_')
    {
        no_of_blocks++;
    }
    i++;
}                                                                    280

/* Anfordern von Speicher für die Geheimtextblöcke */
mpztemp2_p = (mpz_t *) VirtualAlloc(NULL, no_of_blocks * sizeof (mpz_t),\
                                    MEM_COMMIT, PAGE_READWRITE);
```

```
/* Umwandlung des Geheimtextstrings in die Geheimtextblöcke */
j = 0;
for (i = 0; i < no_of_blocks; i++)
{
    str_len = 0;
    while (buf_p[j+str_len] != '_')
    {
        str_len++;
    }
    buf_p[j+str_len] = '\0';
    mpz_init(mpztemp2_p[i]);
    mpz_set_str(mpztemp2_p[i], &buf_p[j], BASE);
    j += str_len + 1;
}

/* Freigeben des Speichers des Geheimtextstrings */
VirtualFree(buf_p, 0, MEM_RELEASE);

/* Geheimtextblöcke mit sk entschlüsseln */
no_of_blocks /= 2;
mpztemp_p = decrypt(mpztemp2_p, sk, no_of_blocks);

/* Freigeben des Speichers der Geheimtextblöcke */
for (i = 0; i < 2 * no_of_blocks; i++)
{
    mpz_clear(mpztemp2_p[i]);
}
VirtualFree(mpztemp2_p, 0, MEM_RELEASE);

/* Klartextblöcke zum Klartextstring zusammensetzen */
buf_p = arr2str(no_of_blocks, mpztemp_p, L-1);

/* Freigeben des Speichers der Klartextblöcke */
for (i = 0; i < no_of_blocks; i++)
{
    mpz_clear(mpztemp_p[i]);
}
VirtualFree(mpztemp_p, 0, MEM_RELEASE);

/* Ausgabe des Klartextstrings */
SetDlgItemText(hwnd, IDE_Klartext, buf_p);

/* Freigabe des Speichers des Klartextstrings */
VirtualFree(buf_p, 0, MEM_RELEASE);
break;


case IDB_SPK:
ZeroMemory(&sfn, sizeof (OPENFILENAME));

lstrcpy(szbuf, "public_key");
sfn.lStructSize = sizeof (OPENFILENAME);
sfn.hwndOwner = hwnd;
```

```
sfn.lpstrFilter = "Public-Key (*.txt)\0*.txt\0";
sfn.lpstrFile = szbuf;                                                    340
sfn.nMaxFile = sizeof (szbuf);
sfn.Flags = OFN_EXPLORER | OFN_HIDEREADONLY |\
            OFN_OVERWRITEPROMPT | OFN_PATHMUSTEXIST;
sfn.lpstrDefExt = "pk";

if (!GetSaveFileName(&sfn))
{
    break;
}
                                                                          350
hFile = CreateFile(szbuf, GENERIC_WRITE, FILE_SHARE_WRITE, NULL,
                   CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if (hFile == INVALID_HANDLE_VALUE)
{
    break;
}

/* Anfordern von Speicher für den Ausgabestring */
buf_p = (char *) VirtualAlloc(NULL, 4 * MAX_BIT * sizeof (char),\
                              MEM_COMMIT, PAGE_READWRITE);                360

/* Schlüsselparameter auslesen */
lstrcpy(buf_p, "p      = ");
GetDlgItemText(hwnd, IDE_p, szbuf, sizeof (szbuf));
lstrcat(buf_p, szbuf);

lstrcat(buf_p, "\r\nalpha = ");
GetDlgItemText(hwnd, IDE_alpha, szbuf, sizeof (szbuf));
lstrcat(buf_p, szbuf);
                                                                          370
lstrcat(buf_p, "\r\nbeta  = ");
GetDlgItemText(hwnd, IDE_beta, szbuf, sizeof (szbuf));
lstrcat(buf_p, szbuf);

/* Schlüsselparameter in Datei schreiben */
WriteFile (hFile, buf_p, lstrlen(buf_p), &dwBytes, NULL);

/* Datei schließen und Speicher freigeben */
CloseHandle (hFile);
VirtualFree(buf_p, 0, MEM_RELEASE);                                       380

break;


case IDB_SGT:
ZeroMemory(&sfn, sizeof (OPENFILENAME));

lstrcpy(szbuf, "geheim");
sfn.lStructSize = sizeof (OPENFILENAME);
sfn.hwndOwner = hwnd;                                                     390
sfn.lpstrFilter = "Geheimtext (*.txt)\0*.txt\0";
sfn.lpstrFile = szbuf;
```

```
        sfn.nMaxFile = sizeof (szbuf);
        sfn.Flags = OFN_EXPLORER | OFN_HIDEREADONLY |\
                    OFN_OVERWRITEPROMPT | OFN_PATHMUSTEXIST;
        sfn.lpstrDefExt = "txt";

        if (!GetSaveFileName(&sfn))
        {
            break;                                                          400
        }

        hFile = CreateFile(szbuf, GENERIC_WRITE, FILE_SHARE_WRITE, NULL,
                           CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
        if (hFile == INVALID_HANDLE_VALUE)
        {
            break;
        }

        /* Bestimmung der Länge des Geheimtextes */                         410
        str_len = SendDlgItemMessage(hwnd, IDE_Geheimtext, WM_GETTEXTLENGTH, 0, 0);

        /* Anfordern von Speicher für den Geheimtextstring */
        buf_p = (char *) VirtualAlloc(NULL, ++str_len * sizeof (char),\
                                      MEM_COMMIT, PAGE_READWRITE);

        /* Geheimtext aus Editbox auslesen */
        GetDlgItemText(hwnd, IDE_Geheimtext, buf_p, str_len);

        /* Geheimtext in Datei schreiben */                                 420
        WriteFile (hFile, buf_p, str_len, &dwBytes, NULL);

        /* Datei schließen und Speicher freigeben */
        CloseHandle (hFile);
        VirtualFree(buf_p, 0, MEM_RELEASE);

        break;


        case IDCANCEL:                                                      430
        mpz_clear(pk.p);
        mpz_clear(pk.alpha);
        mpz_clear(pk.beta);
        mpz_clear(sk.p);
        mpz_clear(sk.a);

        EndDialog(hwnd, 0);
        break;
    }
}                                                                           440


void
Cls_OnSysCommand(HWND hwnd, UINT cmd, int x, int y)
{
```

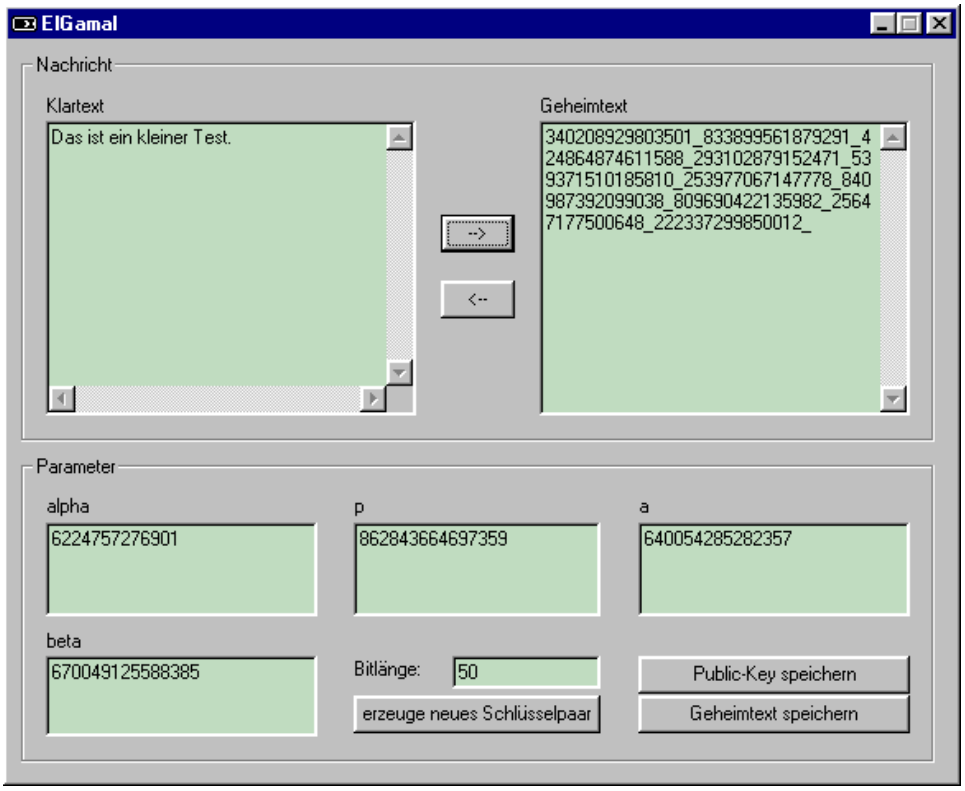

Abbildung 1: Ansicht des Benutzerinterface

```
    switch (LOWORD (cmd))
    {
        case SC_CLOSE:
        mpz_clear(pk.p);
        mpz_clear(pk.alpha);
        mpz_clear(pk.beta);
        mpz_clear(sk.p);
        mpz_clear(sk.a);

        EndDialog(hwnd, 0);
    }
}
```

3 Schlußbetrachtungen

Mit dem vorgestellten Programm ist es möglich, unter MS Windows Zeichenketten nach dem ELGAMAL-Verfahren zu chiffrieren und dechiffrieren. Außerdem kann ein geeignetes Schlüsselpaar erzeugt werden.

Die Daten werden grundsätzlich manuell in die Editboxen eingetragen. Zusätzlich können der Geheimtext sowie die Parameter des öffentlichen Schlüssels abge-

speichert werden. In der Praxis würde man auch eine entsprechende Dateieingabe vorsehen und den privaten Schlüssel dann nochmals mit einem Paßwort schützen. Für die Demonstration der Chiffrierung reicht diese Anwendung jedoch aus.

Für Ende März 2000 ist eine stark erweiterte und verbesserte GMP 3.0 geplant. Z. B. soll die Multiplikation dann mindestens doppelt so schnell sein, die modulare Inversion mindestens fünfmal schneller. Neue Funktionen unter anderem zur Bestimmung von Primzahlen (`mpz_next_prime`) und für das kleinste gemeinsame Vielfache (`mpz_lcm`) werden verfügbar sein. Dadurch ließe sich auch die Bitlänge des Schlüssel vergrößern, ohne daß lange Rechenzeiten entstehen. Die Operandengrößen sollen auch nicht mehr in `limbs`[3] angegeben werden, sondern in `bit`.

Der Quellkode und das Programm unterliegen der GNU General Public License Version 2 oder höher (`http://www.gnu.org/`).

A Anhang

Im folgenden eine Übersicht über einige Bibliotheken für Zahlenarithmetik und Kryptographie.

freelip `ftp://ftp.ox.ac.uk/pub/math/freelip/`, C, frei für Lehr- und Forschungszwecke, Arithmetik für große Zahlen

GMP `http://www.swox.com/gmp/`, C, GPL, Arithmetik für große Zahlen

Miracl `http://indigo.ie/~mscott/`, C/C++, frei für nichtkommerzielle Zwecke, Arithmetik für große Zahlen

RSAEuro C, frei für nichtkommerzielle Zwecke, Kryptofunktionen

Crypto++ `http://www.eskimo.com/~weidai/cryptlib.html`, C++, überwiegend public domain, Kryptofunktionen

cryptlib `http://www.cs.auckland.ac.nz/~pgut001/cryptlib/`, C, frei für nichtkommerzielle Zwecke, kryptographischer Werkzeugkasten

[3] abhängig von Architektur, z. B. 32 bit