Grundlagen der Kryptographie
Anhang Krypto-Interpreter

Biljana Cubaleska Andrea Frank Bernhard Löhlein
Nour-Eddine Ourahou Sonja Schaup
Oliver Stutzke

Lehrgebiet Kommunikationssysteme/FTK
Fachbereich Elektrotechnik
FernUniversität Hagen

13. Oktober 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Der Krypto-Interpreter

Bei diesem Programm handelt es sich um einen Interpreter für kryptologische Protokolle, der mit SDL-PR (*Phrase Representation*) Syntax arbeitet und diese einerseits ausführt und andererseits in SDL-GR (*Graphic Representation*) umsetzt. SDL (*Specification and Description Language*) wurde von ITU-T (*International Telecommunication Union, Telecommunication Standardization*), ehemals CCITT, standardisiert und erlaubt die eindeutige und unmißverständliche Spezifikation und Beschreibung von Kommunikationsprotokollen.

In erster Linie wurde die Syntax von SDL-Prozessen umgesetzt und durch viele Funktionen und Prozeduren aus der Kryptologie ergänzt. Die Prozesskommunikation kann - je nach gewählter Sitzung - entweder nur lokal, direkt oder indirekt über Dateien, über DDE oder auch mit Hilfe von TCP/IP stattfinden.

Im Bereich der Kryptologie stehen neben mathematischen Funktionen (modulare Exponentiation, Primzahltests und einfache Faktorisierungsalgorithmen) und Zufallsfunktionen viele Funktionen im Bereich Verschlüsselungsfunktionen (DES, IDEA, RC5), Schieberegisterfunktionen und Hashfunktionen (Squaremod, MD5, SHA1) zur Verfügung. Bei der Implementierung wurde großer Wert auf Ausführungsgeschwindigkeit gelegt, so dass die Funktionen auch mit hohen Zahlen (z. B. 512 bis 1024 Bit) in angemessener Zeit arbeiten.

Der Interpreter wurde und wird auch weiterhin erweitert, so dass - neben der Kryptologie - zusätzliche Funktionen und Prozeduren aus den Bereichen der Codierungstheorie und der Theorie endlicher Körper (Polynomfunktionen) integriert werden. Außerdem stehen einige Netzfunktionen zur Verfügung.

Der Interpreter wurde für Windows 3.1 entwickelt, inzwischen existiert aber auch eine 32-Bit-Version mit echtem Multithreading für Windows 95 und Windows NT.

Der Interpreter wurde von Andreas Rieke an der FernUni Hagen, Fachgebiet Kommunikationssysteme (Prof. Dr.-Ing. F. Kaderali), Feithstr. 142, 58084 Hagen, Deutschland entwickelt. Anregungen bzw. Kritik per Email an

`andreas.rieke@isl-online.de`

oder per Post sind ausdrücklich erwünscht. Dr.-Ing. Andreas Rieke ist seit 1999 Geschäftsführer der Firma ISL Internet Sicherheitslösungen GmbH, Feithstr. 142, 58097 Hagen.

Der Krypto-Interpreter wird in einer Online und Offline Version angeboten. Installation und Bedienung dieser zwei Varianten werden in den nächsten Abschnitten beschrieben.

A.1 Installation der Offline Version

A.1.1 Windows 3.1

Die Installation unter Windows 3.1 erfolgt durch folgende Schritte:

1. Legen Sie die CD „Kryptologie: Technischer Datenschutz in Kommunikationsnetzen“ ein.

2. Starten Sie Windows, wechseln Sie in den Programmanager und wählen Sie „Ausführen“ aus dem Menü „Datei“.

3. Geben Sie in der Befehlszeile je nach benutztem Laufwerk
   `x:\interpre\install.exe`
   ein, wobei x den Laufwerksbuchstaben bezeichnet, den Ihr CD-ROM-Laufwerk hat.

4. Wählen Sie in dem darauffolgenden Fenster das Zielverzeichnis, in das der Interpreter installiert werden soll, die Verzeichnisse für die jeweiligen Schlüssel sowie als Version die 16-Bit-Version. (Die 32-Bit-Version ist nur mit echten 32-Bit-Betriebssystemen, d. h. insbesondere nicht mit WIN32S, lauffähig.) Da es sich um keine großen Datenmengen handelt, wird das Programm komplett auf der Platte installiert und ist dann auch ohne CD lauffähig.

5. Erstellen Sie sich nun im Programmanager mit dem Befehl Datei, Neu, Programm ein Symbol für den Interpreter. Als Befehlszeile tragen Sie hierbei das Verzeichnis, das Sie als Zielverzeichnis gewählt haben, gefolgt von
   `\win_16.ger\crypto.exe`
   ein. Falls Sie die Standardeinstellungen des Zielverzeichnisses nicht geändert haben, lautet die Befehlszeile
   `c:\crypto\win_16.ger\crypto.exe.`

6. Tragen Sie als Arbeitsverzeichnis ebenfalls das Zielverzeichnis, gefolgt von
   `\PROTO,`
   ein.

A.1.2 Windows 95

Die Installation unter Windows 95 erfolgt durch folgende Schritte:

1. Legen Sie die CD „Kryptologie: Technischer Datenschutz in Kommunikationsnetzen“ ein.

2. Starten Sie Windows 95 und wählen Sie aus dem Menü „Start“ den Befehl „Ausführen“.

3. Geben Sie unter „Öffnen“ je nach benutztem Laufwerk
   `x:\interpre\install.exe`
   ein, wobei x den Laufwerksbuchstaben bezeichnet, den Ihr CD-ROM-Laufwerk hat.

4. Wählen Sie in dem darauffolgenden Fenster das Zielverzeichnis, in das der Interpreter installiert werden soll, die Verzeichnisse für die jeweiligen Schlüssel sowie als Version die 32-Bit-Version. (Die 32-Bit-Version ist nur mit echten 32-Bit-Betriebssystemen lauffähig.) Da es sich um keine großen Datenmengen handelt, wird das Programm komplett auf der Platte installiert und ist dann auch ohne CD lauffähig.

5. Erstellen Sie sich nun auf dem Desktop mit Hilfe der rechten Maustaste und dem Befehl Neu, Verknüpfung ein Symbol für den Interpreter. Als Befehlszeile tragen Sie hierbei das Verzeichnis, das Sie als Zielverzeichnis gewählt haben, gefolgt von
   `\win_32.ger\crypto.exe`
   ein. Falls Sie die Standardeinstellungen des Zielverzeichnisses nicht geändert haben, lautet die Befehlszeile
   `c:\crypto\win_32.ger\crypto.exe .`

6. Tragen Sie als Arbeitsverzeichnis ebenfalls das Zielverzeichnis, gefolgt von
   `\PROTO,`
   ein.

7. Falls die CD in einem Netzlaufwerk liegt, weisen Sie diesem Laufwerk einen Laufwerksbuchstaben zu und verfahren Sie dann entsprechend.

A.2 Bedienung der Offline Version

Der Interpreter präsentiert sich in der gewohnten Windows-Oberfläche, insofern ist die Bedienung für den Windows-erfahrenen Anwender kein Problem. Zunächst wird beispielhaft die Ausführung eines Protokolls beschrieben, bevor im folgenden die Programmelemente besprochen werden, die vom Windows-Standard abweichen oder spezifisch für den Interpreter sind. Eine exakte Beschreibung der Menübefehle finden Sie in der Hilfe zum Programm.

A.2.1 Erste Schritte mit dem Interpreter

Als erstes Beispiel wird im folgenden die Verschlüsselung und anschließende Entschlüsselung nach dem RSA-Protokoll beschrieben. Aus Gründen der Einfachheit werden dazu mitgelieferte Schlüssel benutzt. Sie können Ihre eigenen Schlüssel später berechnen. Starten Sie den Interpreter und wählen Sie „Datei-Öffnen“ aus dem Menü. Suchen Sie die Datei

`..\PROTO\RSA\SEND.PRO`

und laden Sie diese. Klicken Sie nun auf das rechte Fenster und laden Sie dort das Protokoll `RECEIVE.PRO` aus dem gleichen Verzeichnis. Sie sehen jetzt im linken oberen Fenster das RSA-Sendeprotokoll in Textform, während das entsprechende Empfangsprotokoll im Fenster rechts oben dargestellt wird. Wählen Sie „Interpreter/Alle interpretieren“ aus dem Menü. Beide Protokolle werden jetzt parallel interpretiert und in die entsprechende Grafik umgesetzt. Der Sendeprozess beendet sich selbst, nachdem die Nachricht versandt wurde, während der Empfangsprozess die empfangene Nachricht entschlüsselt und ausgibt. Aktivieren Sie jetzt eines der unteren Fenster. Dort wurde das Protokoll in der SDL-GR (Grafik)-Version ausgegeben, das Fenster ist jedoch zu klein, um die Grafik komplett darzustellen. Vergrößern Sie das Fenster und verkleinern Sie die Grafik (im Menü „Protokoll/Symbole kleiner“) solange, bis Sie die komplette Grafik sehen können.

A.2.2 Sitzungen und Kommunikationsarten

Die Kommunikation zwischen mehreren Prozessen und Programmen ist eines der wichtigsten Elemente von SDL. Als Programm wird hier der Krypto-Interpreter selbst verstanden, der ja mehrfach auf einem Rechner oder in Rechnernetzen laufen kann, während Prozesse in den einzelnen Anwenderfenstern innerhalb eines Programms laufen. Um verschiedene Kommunikationsmöglichkeiten zu nutzen, wurden im Interpreter die folgenden Sitzungstypen benutzt:

- Lokale Sitzungen:

  In lokalen Sitzungen arbeiten alle Anwender lokal mit dem Interpreter. Es gibt keine Kommunikation zu anderen Programmen. Nach der Wahl des Menübefehls erscheint ein Dialog, der die Wahl zwischen einem oder zwei Anwendern ermöglicht und deren Namen verlangt. Sollten in einem Prozess Signale an die Umgebung versendet werden, können diese durch eine Option auf die lokalen Prozesse umgeleitet werden.

- DDE:

  Hier ist zusätzlich die Kommunikation zu anderen Programmen auf dem gleichen Rechner möglich. Normalerweise werden Signale nur lokal versandt; Signale an die Umgebung werden an das andere Programm weitergereicht. Durch das Setzen der Optionen im Dialog kann dieses Verhalten beeinflußt werden.

- Direkte Remote-Sitzungen:

  In direkten Remote-Sitzungen wird die Kommunikation mit anderen Programmen über Dateien ermöglicht. Ein Nachrichtenaustausch kann nur erfolgen, wenn beide Programme gleichzeitig aktiv sind. Dazu muss ein Verzeichnis angegeben werden, das für den Austausch von Nachrichten geeignet ist (d. h. alle Anwender müssen sowohl Lese- als auch Schreibrechte in diesem Verzeichnis haben). Auf dieses Verzeichnis müssen sich alle Anwender einigen.

- Indirekte Remote-Sitzungen:

  Hier besteht die Möglichkeit, eine Kommunikation zwischen zwei Anwendern aufzubauen, die nicht gleichzeitig aktiv sind. Dazu muss der sendende Anwender als erster diese Sitzungsart wählen; er kann die gleichen Einstellungen wie bei der direkten Version benutzen. Die Nachrichten können z. B. auf einer Diskette abgelegt werden. Wenn der Empfänger diese Diskette erhält und eine indirekte Remote-Sitzung damit aufbaut, erhält er alle Nachrichten, die der Absender darauf untergebracht hat.

- TCP/IP-Sitzungen:

  Mit Hilfe des Internet-Protokolls können Verbindungen zu anderen Rechnern aufgebaut werden. Auch kann angegeben werden, ob Verbindungen von anderen Rechnern angenommen werden sollen und falls ja, auf welchem Port.

- TCP/IP-Server-Sitzungen:

  Für jede ankommende Verbindung wird ein eigenes Interpreter-Fenster aufgebaut, das zunächst keinerlei Kommunikation außerhalb der mit dem Partner bietet (insbesondere werden keine lokalen Signale ausgetauscht). Wenn die Verbindung beendet wird, kann das Fenster wieder geschlossen werden.

A.3 Bedienung der Online-Version

Sie finden den online Krypto-Interpreter im Internet unter dem Link
`http://www.et-online.fernuni-hagen.de/rub/lehre`

A.3.1 Betriebsmodi

Der Krypto-Interpreter kann in zwei verschiedenen Betriebsmodi arbeiten. Man kann entweder einzelne Ausdrücke bzw. ganze Protokollabläufe vom Krypto-Interpreter auswerten lassen.

Bei der Auswertung von einzelnen Ausdrücken können Ausgaben einzelner Anweisungen oder Funktionen erzeugt werden. In diesem Betriebsmodus des Krypto-Interpreter wird ein sogenannter History-Mechanismus zur Verfügung gestellt, der die Möglichkeit bietet, zu dem Ergebnis des

aktuellen Ausdrucks auch alle Ergebnisse der vorherigen Schritte auf einen Blick zu betrachten.

Dieser Modus ist jedoch wenig geeignet für ganze Protokollabläufe, bei denen eine Folge von Ausdrücken berechnet wird. Dabei können auch bereits berechnete Audrücke in weiteren Anweisungen wieder verwendet werden.

Ein wesentlicher Unterschied der beiden Modi besteht in der Ausgabe. Wird bei einzelnen Ausdrücken das Ergebnis eines jeden Ausdrucks angezeigt, so müssen während eines Protokollablaufs Ausgaben durch Aufruf der Funktion `writeln` erzeugt werden.

A.3.2 Darstellung

Beim online Krypto-Interpreter wird das Fenster Ihres Web-Browsers in vier vorerst gleich große Ausschnitte unterteilt: die Eingabe, die Ausgabe, die Syntax und die Hilfe (siehe Abbilung A.1).

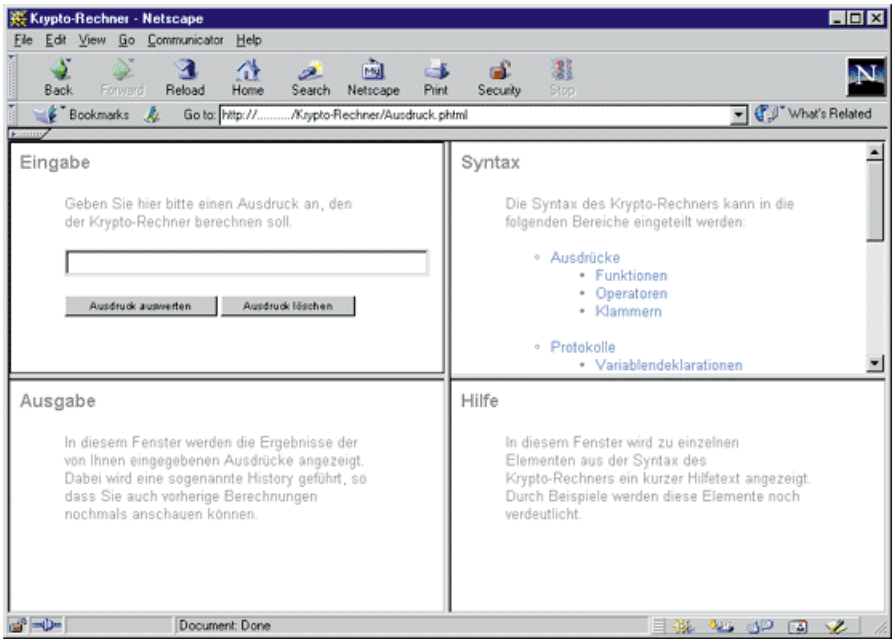

Abbildung A.1: Darstellung des online Krypto-Interpreter.

Mit Hilfe der Maus kann man die Größe der einzelnen Ausschnitte auch durch Ziehen der Trennlinien zwischen den einzelnen Ausschnitten verändern. Die einzelnen Ausschnitte werden im folgenden näher beschrieben.

A.3.3 Eingabe

Abhängig von der Art der gewünschten Berechnung wird die Eingabe entweder in einem einzeiligen Textfeld für einzelne Ausdrücke oder einem mehrzeiligen Textfeld für ganze Protokollabläufe dargestellt (siehe Abbilung A.2).

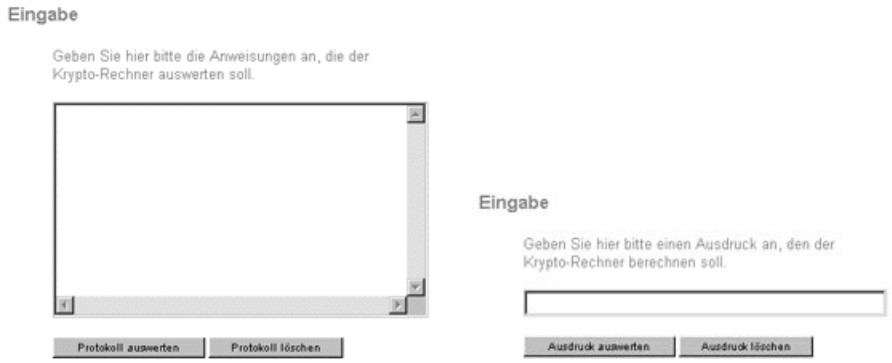

Abbildung A.2: Online-Eingabe eines Ausdrucks (links) und Eingabe eines Protokolls.

Durch Betätigen der Schaltfläche „Ausdruck auswerten“ bzw. „Protokoll auswerten“ wird die Berechnung gestartet und das Ergebnis in der Ausgabe angezeigt.

A.3.4 Ausgabe

In der Ausgabe werden die Ergebnisse der zu berechnenden Ausdrücke bzw. Protokollabläufe angezeigt.

Wurden mehrere Audrücke nacheinander ausgewertet, wird in der Ausgabe eine sogenannte History geführt, die jedoch durch Betätigen der Schaltfläche „Ausgabe löschen“ gelöscht werden kann (siehe Abbilung A.3).

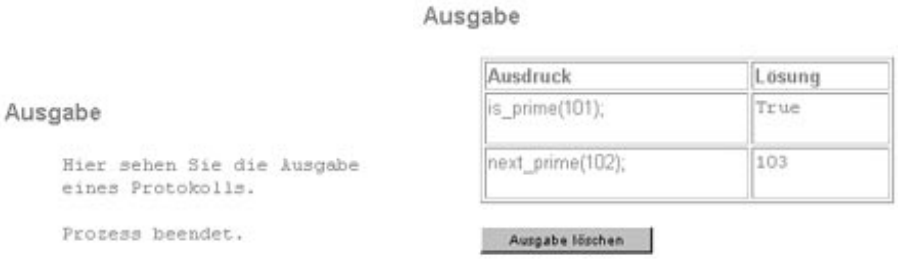

Abbildung A.3: Ausgabe der ausgewerteten Ausdrücke (links) und Protokolle.

A.3.5 Syntax

In dem Ausschnitt oben rechts wird die Syntax des Krypto-Interpreter beschrieben. Die Syntax gibt einen guten Überblick über die Möglichkeiten, die man bei der Benutzung des Krypto-Interpreter hat. Zu einzelnen Elementen der Syntax wird in der Hilfe ein zusätzlicher Hilfetext sowie Beispiele, die Ihnen die Bedienung veranschaulichen sollen, angezeigt.

Außerdem gibt es in diesem Ausschnitt Links, über die man immer wieder zur Startseite des Krypto-Interpreter gelangen kann (siehe Abbilung A.4).

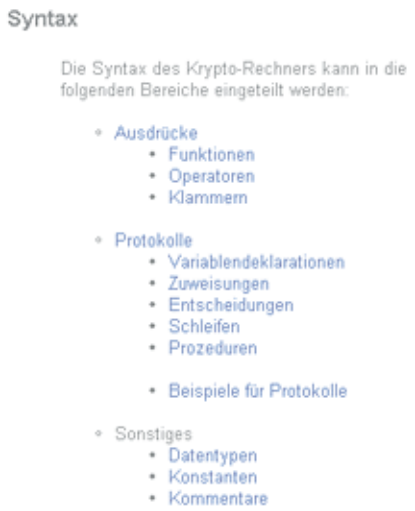

Abbildung A.4: Online-Beschreibung der Syntax des Krypto-Interpreter.

A.3.6 Hilfe

In dem Ausschnitt unten rechts wird zu einzelnen Elementen wie beispielsweise Funktionen oder Prozeduren aus der Syntax des Krypto-Interpreter ein kurzer Hilfetext angezeigt. Zu allen Elementen existieren Beispiele, die diese Elemente noch verdeutlichen. Durch Anklicken der entsprechenden Links werden die Ergebnisse dieser Beispiele in der Ausgabe angezeigt (siehe Abbilung A.5).

Abbildung A.5: Online-Hilfe des Krypto-Interpreter.

???

A.4 Die Syntax

Die SDL-Syntax wurde in Anlehnung an Pascal entwickelt und weist lediglich bei der Kommunikation verschiedener Prozesse wesentliche Änderungen auf. Die Syntax ist ausführlich und komplett in der Online-Hilfe beschrieben, im folgenden soll es lediglich darum gehen, einen kurzen Überblick über die wesentlichen Bereiche zu geben.

A.4.1 Ausdrücke

Der Interpreter kann Ausdrücke einzeln oder in Anweisungen verarbeiten. Einzelne Ausdrücke können in einer Zeile eingegeben werden. Bei Verwendung von Klammern (z. B. runde öffnende Klammer in der 1. Zeile und entsprechende schließende Klammer in der 2. Zeile) können sich Ausdrücke auch über mehrere Zeilen aufteilen. Ausdrücke müssen durch ein Semikolon abgeschlossen werden und können aus den folgenden Elementen bestehen:

- Konstanten
- Variablen
- Operatoren
- Klammern
- Funktionen

Beispiele für Ausdrücke sind:

```
44 + 55;
```

oder

```
load ( 'c:\test.num' );
```

Konstanten können in allen verfügbaren Datentypen eingegeben werden. Integer-Konstanten können in dezimal oder sedezimal (durch führendes 0x oder $ gekennzeichnet) eingegeben werden, für Boolean-Konstanten ist „`True`“ und „`False`“ definiert. String-Konstanten (Datentyp „`Charstring`“) müssen durch einfache Anführungszeichen eingeschlossen werden. Real-Konstanten müssen, um von Integer-Konstanten unterschieden werden zu können, einen Nachkommaanteil haben. Der Interpreter verwendet zusätzlich den Datentyp PID (Prozess-Id) und selbstdefinierte Datentypen (Arrays und Strukturen).

Beispiele:

```
1234
0x123456789ABCDEF
$FEDCBA987654321
'Dies ist ein String'
False
```

Variablen können für alle verwendeten Datentypen benutzt werden. Die folgende Tabelle zeigt die im Interpreter implementierten Operatoren. Die Abkürzungen B, I, P, R und S stehen für die entsprechenden Datentypen `Boolean`, `Integer`, `PID`, `Real` und `Charstring`.

Die im Krypto-Interpreter implementierten Operatoren, Funktionen und Prozeduren sind in Gruppen aufgeteilt und werden in den folgenden Tabellen beschrieben:

- Operatoren in Tabelle A.1,
- Dateifunktionen in Tabelle A.2,
- mathematische Funktionen in den Tabellen A.3 und A.4,
- Zufallsfunktionen in Tabelle A.5,
- Umwandlungsfunktionen in Tabelle A.6,
- Codierungsfunktionen in Tabelle A.7,
- Verschlüsselungsfunktionen in Tabelle A.8,
- Schieberegisterfunktionen in Tabelle A.9,
- Polynomfunktionen in Tabelle A.10,
- Hashfunktionen in Tabelle A.11,
- Netzfunktionen in Tabelle A.12,
- sonstige Funktionen in Tabelle A.13,
- Prozeduren in Tabelle A.14.

In den Tabellen werden mit den Abkürzungen B, I, P, R und S die verwendeten Datentypen (X bezeichnet einen beliebigen Datentyp) und mit den Namen A, B, C und D für die Parameter bzw. Z das Ergebnis bezeichnet.

Die Prioritäten der Operatoren können durch runde Klammern modifiziert werden.

Beispiel:

```
( 3 + 4 ) * 5;
```

Beispiele für Funktionen:

```
mod_exp (a,b,c)
load ( ‘f:\test.num‘ )
```

Bei allen Zufallsfunktionen, die mit den Silben `rand_bits` beginnen, gibt der erste Parameter die gewünschte Länge der Zufallszahl (Binärstellen) an. Die Funktionen mit den Silben `rand_max` berechnen eine Zufallszahl $0 < Z < A$. Alle Zufallsfunktionen, die auf die Silbe gcd enden, arbeiten mit der Nebenbedingung $\gcd(B, Z) = 1$. Die Funktionen, die auf prime enden, berechnen zufällige Primzahlen. Im Parameter B kann die Fehlerwahrscheinlichkeit angegeben werden. Falls das nicht geschieht, wird mit einer Fehlerwahrscheinlichkeit von 0.1% gearbeitet.

A.4.2 Anweisungen

Anweisungen können in einer Zeile eingegeben werden; die Eingabe über mehrere Zeilen ist auch möglich, wenn entsprechende Klammersymbole gesetzt werden. Zu den Klammersymbolen zählen neben den runden Klammern auch Schlüsselwörter wie `DECISION` / `ENDDECISION`, `repeat` / `until` und `STATE` / `ENDSTATE`. Anweisungen werden grundsätzlich durch ein Semikolon abgeschlossen. Neben Leeranweisungen sind die folgenden Anweisungen im Interpreter implementiert:

- Variablendeklarationen
- Zuweisungen
- Typdefinitionen
- Entscheidungen
- Prozedurdefinitionen
- Prozeduraufrufe
- Makroaufrufe
- `OUTPUT`-Anweisungen
- Zustände
- `NEXTSTATE`-Anweisungen
- Synonymdefinitionen

Variablendeklarationen können für alle verwendeten Datentypen erfolgen. Eingeleitet wird die Deklaration durch das Schlüsselwort `DCL`, dem die durch Kommata getrennten Variablennamen folgen. Die Angabe des Datentyps (`Boolean`, `Charstring`, `Integer`, `PID`, `Real` oder selbstdefinierter Typ) und das abschließende Semikolon vervollständigen die Deklaration. Ein Variablenname kann nur für einen Datentyp deklariert werden, die doppelte (bzw. mehrfache) Deklaration ist nicht zulässig.

Beispiele:

```
DCL a, b, c integer;
DCL text charstring;
```

Zuweisungen haben die Form

```
TASK Variable := Ausdruck;
```

, wobei Variable und Ausdruck den gleichen Datentyp haben müssen.

Beispiele:

```
TASK a := 7;
TASK bcd := exp ( 8, 9 ) mod 3;
TASK text := 'ABC';
```

Der Interpreter erlaubt die Definition neuer Datentypen in Form von Strukturen und Arrays. Sobald ein Datentyp neu geschaffen wurde, kann er wie die grundlegenden Datentypen benutzt werden. Die Definition von Stukturen ist in dem folgenden Beispiel gezeigt:

```
NEWTYPE t STRUCT
i integer;
c charstring;
ENDNEWTYPE;
```

Die folgenden Beispiele zeigen den Zugriff auf Strukturen:

```
DCL d t;
TASK d ( i ) := 11;
CALL writeln ( d ( i ) );
```

Mit Arrays bietet sich die andere Möglichkeit, eigene Datentypen zu schaffen. Die Definition von Arrays kann dem folgenden Beispiel entnommen werden:

```
NEWTYPE a ARRAY (Integer, 5) ENDNEWTYPE;
```

Der Zugriff auf Arrays ist im Bereich von 1 bis zum Maximalwert (in diesem Beispiel von Index 1 bis 5) möglich:

```
DCL d a;
TASK d ( 3 ) := 11;
CALL writeln ( d ( 3 ) );
```

Die Syntax von Entscheidungen wird wie folgt definiert: Nach dem Schlüsselwort `DECISION` folgt eine Frage (Ausdruck mit Ergebnistyp `BOOLEAN`), die durch ein Semikolon abgeschlossen wird. Als erste Antwort folgt „`(true):`“, gefolgt von Anweisungen. Es kann eine zweite Antwort folgen (muss aber nicht), sie wird durch „`ELSE:`“ eingeleitet, worauf ebenfalls mindestens eine Anweisung folgen muss. Die Entscheidung wird durch das Schlüsselwort `ENDDECISION` abgeschlossen, dem ein Semikolon folgen muss.

Beispiel:

```
DECISION 2 > 3;
  (true):
    OUTPUT test1;
  ELSE:
    OUTPUT test2;
ENDDECISION;
```

Im Interpreter können sowohl eigene Prozeduren definiert werden als auch vordefinierte Prozeduren aufgerufen werden. Eine Prozedurdefinition beginnt mit dem Schlüsselwort `PROCEDURE`, gefolgt von dem Prozedurnamen und einem Semikolon. Falls Parameter übergeben werden sollen, muss in der nächsten Anweisung die komplette Parameterliste deklariert werden. Nach dem Schlüsselwort `FPAR` und dem Schlüsselwort `IN` oder der Folge `IN` / `OUT` (je nachdem, ob der Parameter nur übergeben oder auch zurückgegeben werden soll) folgt eine durch Kommata getrennte Liste von Variablennamen, gefolgt von dem Datentyp. Nach einem Komma können weitere `IN`- oder `IN` / `OUT`-Sequenz folgen; die Anweisung wird durch ein Semikolon abgeschlossen.

Innerhalb der Prozedur sollte das Schlüsselwort `RETURN`, gefolgt von einem Semikolon, auftreten, das einen Rücksprung veranlaßt. Das Ende der Prozedur wird durch `END-PROCEDURE`, ebenfalls mit Semikolon, gekennzeichnet.

Beispiel:

```
PROCEDURE multipliziere;
  FPAR IN a, b integer, IN / OUT c integer;
  TASK c := a * b;
  RETURN;
ENDPROCEDURE;
```

Im Interpreter können sowohl eigene Prozeduren als auch vordefinierte Prozeduren aufgerufen werden. Beim Aufruf selbstdefinierter Prozeduren ist zu beachten, dass die Parameter, die zur Rückgabe von Ergebnissen genutzt werden, im Aufruf Variablen sind. Im folgenden werden vordefinierte Prozeduren beschrieben. In der Tabelle deuten die Abkürzungen B, I, P, R und S auf die entsprechenden Datentypen hin, ein X steht für einen beliebigen Datentyp. Ein * in der Parameterliste bedeutet, dass die Funktion mit beliebig vielen Parametern des entsprechenden Datentyps aufgerufen werden kann. In der Spalte „Beschreibung“ werden die Namen A und B für die Parameter benutzt.

Beispiele:

```
CALL writeln ( ‘Dieser Text erscheint im Anwenderfenster.‘
);
CALL seed ( 123 );
```

Da SDL keine Schleifen kennt, wurden entsprechende Makros entworfen, die Schleifen aus Bedingungen und Sprungbefehlen zusammensetzen. Es wurde die aus Pascal bekannte `REPEAT` / `UNTIL`-Schleife implementiert; zusätzlich steht die `WHILE` / `WEND`-Schleife zur Verfügung.

REPEAT / UNTIL:

Vor den Schleifenanweisungen muss

```
MACRO repeat ( name );
```

stehen, wobei name in das entsprechende Verbindersymbol in der grafischen Darstellung eingesetzt wird. Nach beliebig vielen Schleifenanweisungen muss die Folge

```
MARCO until ( bedingung );
```

folgen, wobei `bedingung` einen Ausdruck vom Typ `Boolean` liefern muss.

WHILE / WEND:

Vor den Schleifenanweisungen muss

```
MACRO while ( bedingung );
```

stehen; `bedingung` muss einen Ausdruck vom Typ `Boolean` liefern. Nach beliebig vielen Anweisungen muss die Sequenz

```
MACRO wend ( name );
```

folgen.

INCLUDE:

Auf die Folge

```
MACRO include
```

muss ein in Klammern eingeschlossener String folgen, abgeschlossen durch ein Semikolon. Anstelle dieses Makros wird die angegebene Datei geladen und interpretiert.

Beispiele:

```
DCL a integer;
MACRO repeat ( Schleife );
  TASK a := a + 1;
MACRO until ( a = 10 );
MACRO include ( ‘C:\TEST.INC‘ );
```

Mit Hilfe der `OUTPUT`-Anweisung können Signale an andere Prozesse versandt werden. Nach dem Schlüsselwort `OUTPUT` wird zunächst ein Signalname eingegeben, auf den eine nichtleere Parameterliste folgen kann. Danach kann, nach dem Schlüsselwort `TO`, eine `PID` angegeben werden, so dass nur der zugehörige Prozess das Signal empfängt. Sonderfälle sind die PIDs `SELF` und `ENV`: Signale an `SELF` sind an den eigenen Prozess adressiert, Signale an `ENV` werden von allen Prozessen in der Umgebung empfangen. Andernfalls wird das Signal an alle Prozesse innerhalb des Systems gesandt. Auch die `OUTPUT`-Anweisung wird durch ein Semikolon abgeschlossen. Anstelle des Schlüsselwortes `TO` kann auch das Wort `VIA`, gefolgt von einem Kanalnamen, folgen. Das Signal wird dann nur über den genannten Kanal versandt (funktioniert nur mit TCP/IP).

Beispiele:

```
OUTPUT test;
OUTPUT rsa ( bits ) TO SELF;
```

Eine wesentliche Unterscheidung zu Programmiersprachen wie z.B. Pascal bietet SDL durch Zustände. Ein Prozess befindet sich entweder in einem Zustand oder in einem Zustandsübergang. Um einen Zustand zu verlassen, muss ein bestimmtes Ereignis eintreten. Ein solches Ereignis ist z. B. das Eintreffen eines Signals. Je nach Signal und - falls vorhan-

den - Signalparametern wird eine bestimmte Aktion ausgeführt. Falls das Signal in einem Zustand nicht erwartet wird, wird es verworfen. Andernfalls wird durch das Signal entschieden, welcher Zustandsübergang ausgeführt wird. Nach dem Schlüsselwort `STATE`, dem Zustandsnamen und einem Semikolon muss mindestens eine Eingabe erfolgen. Eine Eingabe beginnt mit dem Schlüsselwort `INPUT` oder `SAVE` gefolgt von dem Signalnamen und evtl. Parametern. Bei `INPUT`-Anweisungen folgt nach einem Semikolon mindestens eine Anweisung. Der Zustand wird durch das Schlüsselwort `ENDSTATE`, gefolgt von einem Semikolon, abgeschlossen. Ein so eingegebener Zustand wird intern abgespeichert und in die entsprechende Grafik umgesetzt. Ausgeführt wird der Zustand erst, wenn ein entsprechender `NEXTSTATE`-Befehl bearbeitet wird. Die `SAVE`-Anweisung bewirkt, dass das Signal nicht gelöscht, sondern gespeichert wird, um evtl. im nächsten Zustand bearbeitet zu werden.

Bei Dash-States (`STATE *;`) werden alle Eingaben in alle anderen Zustände des Prozesses kopiert. In Dash-`NEXTSTATE`-Anweisungen wird jeweils der Zustandsname eingesetzt. Es darf nur ein Dash-`STATE` pro Prozess existieren; nur in diesem sind Dash-`NEXTSTATE`-Anweisungen erlaubt.

Innerhalb von Zuständen enthält die PID-Variable `SENDER` die `PID` des Prozesses, von dem das letzte Signal empfangen wurde.

Beispiel:

```
STATE rsa_mul;
  INPUT rsa_mul_next ( wert );
    TASK message := message // unpack ( mod_exp ( wert, d, n
) );
    NEXTSTATE rsa_mul;
  INPUT rsa_mul_last;
    CALL writeln ( ‘RSA-Mehrfachuebertragung mit ‘, bits, ‘
Bits:‘ );
    CALL writeln ( message );
    NEXTSTATE ready;
  SAVE test;
ENDSTATE;
```

Eine `NEXTSTATE`-Anweisung führt einen Zustand aus, der vorher definiert worden ist. Nach dem Schlüsselwort `NEXTSTATE` wird zunächst ein Zustandsname erwartet, gefolgt von einem Semikolon.

Beispiel:

```
NEXTSTATE test;
```

Synonyme werden im gesamten Prozess oder in allen folgenden Anweisungen durch den bei der Definition angegebenen Wert ersetzt. Eine Synonymdefinition beginnt mit dem Schlüsselwort `SYNONYM`, gefolgt von dem Synonymnamen, einem Gleichheitszeichen, dem Wert und dem abschließenden Semikolon.

Beispiel:

```
SYNONYM pi = 3.141592654;
```

A.4.3 Prozesse

Für komplette Prozesse gelten die folgenden Syntaxregeln:

1. Schlüsselwort `PROCESS`, gefolgt von dem Prozessnamen und einem Semikolon.
2. Beliebig viele Synonymdefinitionen
3. Beliebig viele Variablendeklarationen
4. Beliebig viele Prozedurdefinitionen
5. `START`-Befehl, gefolgt von einem Semikolon.
6. Beliebige Anweisungen, gefolgt durch eine `NEXTSTATE`- oder `STOP`-Anweisung.
7. Beliebig viele Zustände.
8. Schlüsselwort `ENDPROCESS`, gefolgt von einem Semikolon.

Prozesse können in beliebiger Form auf mehrere Zeilen aufgeteilt werden (`ENDPROCESS` und das folgende Semikolon müssen allerdings in einer Zeile stehen).

A.5 Beispielprotokolle

In den folgenden Unterkapiteln werden einige Beispielprotokolle vorgestellt, die sich mit dem Krypto-Interpreter ausführen lassen.

A.5.1 Das RSA-Verfahren

Um mit Hilfe des RSA-Verfahrens ver- bzw. entschlüsseln zu können, müssen zunächst einmal Schlüssel vorhanden sein. Anstatt die mitgelieferten Schlüssel, die sich im Verzeichnis `KEYS` befinden, zu benutzen, können Sie sich auch selbst Schlüssel berechnen. Laden Sie dazu das Protokoll

`PROTO\RSA\KEYS.PRO.`

Tragen Sie Ihren Namen als Anwender ein (je nach Betriebssystem evtl. mit nur 8 Zeichen) und führen Sie das Protokoll aus. Die Berechnung der Primzahlen kann - je nach Rechner - einige Minuten dauern. Sobald alle Berechnungen abgeschlossen sind, werden die Schlüssel in Dateien gespeichert. Für jeden Anwender können mehrere Schlüssel unterschiedlicher Länge berechnet werden. Die Protokolle `SEND` und `RECEIVE` im gleichen Ordner dienen der Ver- bzw. Entschlüsselung. Im Sendeprotokoll wird die Schlüssellänge festgelegt, die nur dann geändert werden sollte, wenn Sie zu der angegebenen Schlüssellänge auch passende Schlüssel berechnet haben. Dann werden die öffentlichen Schlüssel des Partners geladen, die

zu verschlüsselnde Nachricht wird blockweise verschlüsselt und die verschlüsselten Blöcke werden übertragen, bis nach dem letzten Block das Signal `rsa_last` das Ende der Nachricht signalisiert. Auf der Empfangsseite wird zunächst die vom Sender bestimmte Schlüssellänge empfangen, um danach die passenden Schlüssel zu laden. Die Nachricht wird dann blockweise entschlüsselt und zu einer Gesamtnachricht zusammengesetzt. Sobald das abschließende Signal rsa_last empfangen wird, wird die Nachricht ausgegeben und dann auf die nächste Nachricht gewartet.

A.5.2 Der Diffie-Hellman-Schlüsselaustausch

Protokolle zum Diffie-Hellman-Schlüsselaustausch sind im Ordner

`PROTO\DIF_HEL`

abgelegt. Auch hier besteht der erste Schritt darin, einen eigenen gemeinsam genutzten Schlüssel zu berechnen. Zu diesem Zweck kann das Protokoll `KEYS.PRO` benutzt werden. Laden Sie das Protokoll und tragen Sie die gewünschte Schlüssellänge sowie den Pfad für die Schlüssel ein. Führen Sie das Protokoll anschließend aus; die Schlüssel werden dann berechnet und als Dateien gespeichert. Die Protokolle `ACTIVE.PRO` und `PASSIVE.PRO` dienen zur Durchführung des Schlüsselaustauschs. Das erstgenannte Protokoll initiiert den Schlüsselaustausch, indem es die Schlüssel bestimmt, den ersten Wert berechnet und an den Partner übersendet. Der Partner lädt die gleichen Schlüssel, berechnet und übermittelt seinen Wert. Anschließend berechnen beide Partner den Schlüssel und geben diese auf dem Bildschirm aus. Um die übertragenen Signale wirklich zu sehen, können Sie im Menü „Sitzung“ einen weiteren Anwender einfügen, z. B. mit dem Namen „Angreifer“. Wenn Sie dann alle Protokolle parallel ausführen, sehen Sie im Angreiferfenster die drei ausgetauschten Signale. Dabei fällt insbesondere auf, dass jeder Prozess eine eigene Prozess-Nummer (Prozess-Id, PID) besitzt, die mit jedem Signal als Absenderadresse mitgeschickt wird.

A.5.3 Das Fiat-Shamir-Protokoll

Beim Fiat-Shamir-Protokoll wird die Schlüsselberechnung in zwei Schritten durchgeführt. Die gemeinsam genutzen Schlüssel werden mit dem Protokoll

`FIA_SHA\KEYS\COMMON.PRO`

berechnet, während die Anwenderschlüssel in

`FIA_SHA\KEYS\USER.PRO`

berechnet werden. Die Authentifikation wird mit einem Prüfling (Prover) und einem Prüfer (Verifier) durchgeführt. Der Prüfer bestimmt dabei die verwendeten Schlüssel und die Anzahl der Runden.

Operator	Argument(e)	Ergebnis	Beschreibung
-	I	I	Unäres Minus
-	R	R	Unäres Minus
not	B	B	Logische Negation
*	I / I	I	Multiplikation
*	R / R	R	Multiplikation
/	I / I	I	Integer-Division (div)
/	R / R	R	Division
mod	I / I	I	Integer-Division (mod)
+	I / I	I	Addition
+	R / R	R	Addition
-	I / I	I	Subtraktion
-	R / R	R	Subtraktion
//	S / S	S	Zusammenfügen
=	I / I	B	Gleichheitsoperator
=	P / P	B	Gleichheitsoperator
=	R / R	B	Gleichheitsoperator
=	S / S	B	Gleichheitsoperator
/=	I / I	B	Ungleichheitsoperator
/=	P / P	B	Ungleichheitsoperator
/=	R / R	B	Ungleichheitsoperator
/=	S / S	B	Ungleichheitsoperator
>	I / I	B	Relationaler Operator
>	R / R	B	Relationaler Operator
>=	I / I	B	Relationaler Operator
>=	R / R	B	Relationaler Operator
<	I / I	B	Relationaler Operator
<	R / R	B	Relationaler Operator
<=	I / I	B	Relationaler Operator
<=	R / R	B	Relationaler Operator
and	I / I	I	Bitweise UND-Verknüpfung
and	B / B	B	Logische UND-Verknüpfung
or	I / I	I	Bitweise ODER-Verknüpfung
or	B / B	B	Logische ODER-Verknüpfung
xor	I / I	I	Exklusive bitweise ODER-Verknüpfung
xor	B / B	B	Exklusive logische ODER-Verknüpfung

Tabelle A.1: Operatoren, nach Priorität sortiert (das unäre Minus hat die höchste Priorität).

Funktion	Parameter	Ergebnis	Beschreibung
`load`	S	X	Lädt die Daten aus der angegebenen Datei und gibt den entsprechenden Datentyp zurück.
`load_int`	S	I	Lädt die Zahl aus der angegebenen Datei.
`load_str`	S	S	Lädt einen String aus der angegebenen Datei.
`exist`	S	B	Stellt fest, ob die angegebene Datei existiert.
`lines_in_file`	S	I	Bestimmt die Anzahl der Zeilen in der angegebenen Datei.
`load_str_from_line`	S / I	S	Lädt die Zeile B aus Datei A als String.
`load_int_from_line`	S / I	I	Lädt die Zeile B aus Datei A als Integer.

Tabelle A.2: Dateifunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`mod_mul`	I / I / I	I	Berechnet $Z = (A \cdot B) \bmod C$.
`mod_exp`	I / I / I	I	Berechnet $Z = (A^B) \bmod C$.
`gcd`	I / I	I	Berechnet den größten gemeinsamen Teiler von A und B.
`rcp`	I / I	I	Berechnet den modularen Kehrwert von A zu B.
`is_prime`	I (/ R)	B	Stellt fest, ob A eine Primzahl ist. Dazu kann in B die Fehlerwahrscheinlichkeit angegeben werden (ohne Angabe wird mit 0.1% gerechnet).
`min`	I / I	I	Gibt die kleinere der beiden Zahlen zurück.
`max`	I / I	I	Gibt die größere der beiden Zahlen zurück.
`exp`	I / I	I	Berechnet $Z = A^B$.
`factor`	I	I	Liefert einen Faktor von A. Eine Liste häufig vorkommender Faktoren wird bei einigen Algorithmen in der Datei FACTORS.TXT im Programmverzeichnis durchsucht.
`legendre`	I / I	I	Berechnet die Legendre-Funktion $A^{((B-1)/2)} \bmod B$ für ungerade Primzahlen B. Statt -1 wird $p - 1$ zurückgegeben.
`mod_sqrt`	I / I / I	I	Liefert eine der Quadratwurzeln von $A \bmod B$ oder eine Fehlermeldung, falls keine existiert. Ist nur für ungerade Primzahlen B definiert. Mit $C = 1$ wird die kleinere der beiden Wurzeln zurückgegeben.
`mod_sqrt`	I / I / I / I	I	Liefert eine der Quadratwurzeln von $A \bmod (B \cdot C)$ oder eine Fehlermeldung, falls keine existiert. Ist nur für ungerade und verschiedene Primzahlen B und C definiert. Mit $D = 1$ wird die kleinere der vier Wurzeln zurückgegeben.

Funktion	Parameter	Ergebnis	Beschreibung
`prev_prime`	I (/ R)	I	Berechnet die größte Primzahl kleiner A. Dazu kann in B die Fehlerwahrscheinlichkeit angegeben werden (ohne Angabe wird mit 0.1% gerechnet).
`next_prime`	I (/ R)	I	Berechnet die kleinste Primzahl größer A. Dazu kann in B die Fehlerwahrscheinlichkeit angegeben werden (ohne Angabe wird mit 0.1% gerechnet).
`phi`	I	I	Berechnet die Anzahl der zu A teilerfremden natürlichen Zahlen, die nicht großer als A sind.
`Abs`	I	I	Berechnet den Absolutwert von A.
`Signum`	I	I	Berechnet die Signum-Funktion.
`Lcm`	I	I	Berechnet das kleinste gemeinsame Vielfache.
`Factorial`	I	I	Berechnet die Fakultät von A.
`Binomial`	I / I	I	Berechnet $A!/B!/(A-B)!$.
`Log`	I / I	I	Berechnet den „normalen“ Logarithmus von A zur Basis B. Nachkommastellen werden abgeschnitten.
`Rot`	I / I	I	Zieht die Bte Wurzel von A. Nachkommastellen werden abgeschnitten.
`Pow`	R / R	R	Exponentiation $Z = A^B$.

Tabelle A.4: Mathematische Funktionen (Teil 2).

Funktion	Parameter	Ergebnis
`rand_bits`	I	I
`rand_max`	I	I
`rand_bits_gcd`	I / I	I
`rand_max_gcd`	I / I	I
`rand_bits_prime`	I (/ R)	I
`rand_max_prime`	I (/ R)	I

Tabelle A.5: Zufallsfunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`str2int`	S	I	Wandelt einen String in einen Integer um.
`int2str`	I	S	Wandelt einen Integer in einen String um.
`pack`	S (/ I / I)	I	Wandelt A in einen möglichst kleinen Integer um. Wenn B und C ebenfalls angegeben werden, wird der B. Block mit C Bits Länge umgewandelt, wobei alle Blöcke ein Vielfaches von 8 als Länge haben.
`unpack`	I	S	Rückwandlung von pack.
`hex`	I	S	Wandelt einen Integer in einen String in Sedezimaldarstellung um.
`bin`	I	S	Wandelt einen Integer in einen String in Binärdarstellung um.
`dec`	I	S	Wandelt einen Integer in einen String in Dezimaldarstellung um.
`float`	I	R	Wandelt einen Integer in einen Real-Wert um.
`fix`	R	I	Wandelt einen REAL durch Abrunden in einen Integer um.
`int2poly`	I / S	S	Wandelt den übergebenen Integer A in ein Polynom mit der Variablen B um.
`int2texpoly`	I / S	S	Wandelt den übergebenen Integer A in ein Polynom in TEX-Darstellung mit der Variablen B um.
`oct`	I	S	Wandelt einen Integer in einen String in Oktaldarstellung um.
`convert_to_str`	X	S	Konvertiert den Parameter A in einen String.
`convert_from_str`	S	X	Konvertiert den String A in den entsprechenden Datentyp.

Tabelle A.6: Umwandlungsfunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`cyclic_encode`	I / I	I	Kodiert A mit dem Polynom B und gibt das komplette Kodewort zurück.
`cyclic_syndrome`	I / I	I	Berechnet das Syndrom des Kodewortes A mit dem Polynom B.
`cyclic_decode`	I / I / I / I	I	Dekodiert das Wort A mit dem Polynom B. Die Anzahl der zu korrigierenden Fehler ist in C angegeben; D ist die Länge des Datenbereichs des Kodeworts. Falls das Kodewort mehr Fehler als die angegebenen enthält, wird -1 zurückgegeben.
`cyclic_check`	I / I / I / I	I	Wie `cyclic_decode`, gibt allerdings die Mindestanzahl der gefundenen Fehler zurück.
`cyclic_distance`	I / I	I	Berechnet die Distanz eines zyklischen Codes mit Polynom A für die Blocklänge B. Ist bisher nur für Polynome vom Grad bis zu 32 implementiert.

Tabelle A.7: Codierungsfunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`encrypt_idea`	I / I	I	Verschlüsselt A mit dem Schlüssel B nach dem IDEA-Algorithmus.
`decrypt_idea`	I / I	I	Entschlüsselt A mit dem Schlüssel B nach dem IDEA-Algorithmus.
`encrypt_des`	I / I	I	Verschlüsselt A mit dem Schlüssel B nach dem DES-Algorithmus.
`decrypt_des`	I / I	I	Entschlüsselt A mit dem Schlüssel B nach dem DES-Algorithmus.
`encrypt_rc5`	I / I	I	Verschlüsselt A mit dem Schlüssel B nach dem RC5-Algorithmus.
`decrypt_rc5`	I / I	I	Entschlüsselt A mit dem Schlüssel B nach dem RC5-Algorithmus.

Tabelle A.8: Verschlüsselungsfunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`lfsr_run`	I / I	I	Liefert B Zufallsbits des Registers A.
`lfsr_state`	I	I	Liefert den Zustand des Registers A.

Tabelle A.9: Schieberegisterfunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`is_irreducible`	I	B	Stellt fest, ob das Polynom A irreduzibel ist.
`is_primitive`	I	B	Stellt fest, ob das Polynom A primitiv ist.
`prev_irreducible`	I	I	Berechnet das größte Polynom kleiner A, das irreduzibel ist.
`next_irreducible`	I	I	Berechnet das kleinste Polynom größer A, das irreduzibel ist.
`prev_primitive`	I	I	Berechnet das größte Polynom kleiner A, das primitiv ist.
`next_primitive`	I	I	Berechnet das kleinste Polynom größer A, das primitiv ist.
`poly_mul`	I / I	I	Polynommultiplikation.
`poly_div`	I / I	I	Polynomdivision.
`poly_mod`	I / I	I	Berechnet den Rest einer Polynomdivision.
`poly_mod_exp`	I / I / I	I	Modulare Exponentiation eines Polynoms.
`poly_gcd`	I / I	I	Berechnet den größten gemeinsamen Teiler zweier Polynome.
`poly_lcm`	I / I	I	Berechnet das kleinste gemeinsame Vielfache zweier Polynome.
`poly_factor`	I	I	Liefert einen Faktor des übergebenen Polynoms A zurück. Falls A irreduzibel ist, wird A zurückgegeben. Ist bisher nur für kleine Polynome implementiert.
`poly_order`	I	I	Gibt die Ordnung des übergebenen Polynoms zurück. Ist bisher nur für kleine Polynome implementiert.
`poly_reciprocal`	I	I	Berechnet das zu A reziproke Polynom.
`poly_minimal`	I / I	I	Berechnet das Minimalpolynom zum Polynom A mit dem Erweiterungspolynom B.

Tabelle A.10: Polynomfunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`hash_md5`	S	I	Berechnet die Hashfunktion MD5 für den String A.
`hash_squaremod`	S / I	I	Berechnet die Squaremod-Hashfunktion mit dem Modulus B für den String A.
`hash_sha1`	S	I	Berechnet die Hashfunktion SHA1 für den String A.

Tabelle A.11: Hashfunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`tcpip_connect`	S / I	I	Stellt eine TCP/IP-Verbindung zum Rechner A auf Port B her und gibt eine Verbindungsnummer zurück.
`tcpip_get_remote_addr`	I	I	Gibt die Adresse des Partners der Verbindung A zurück.
`tcpip_get_remote_port`	I	I	Gibt den Port des Partners der Verbindung A zurück.
`tcpip_get_local_addr`		I	Gibt die lokale TCP/IP-Adresse zurück.
`tcpip_get_local_port`	I	I	Gibt den lokalen Port für Verbindung A zurück.

Tabelle A.12: Netzfunktionen.

Funktion	Parameter	Ergebnis	Beschreibung
`digits`	I	I	Gibt die Anzahl der Binärstellen von A zurück.
`receive`		S	Gibt das nächste Signal aus der Warteschlange zurück.
`set_break`	I	I	Setzt die Unterbrechbarkeit des Interpreters. $A = 0$ bedeutet, dass der Interpreter keine Unterbrechungen zulässt und alle Rechenzeit für sich nutzt (kein Multitasking möglich), während $A = 1$ Unterbrechbarkeit und Multitasking zur Folge hat. Der vorherige Zustand wird zurückgegeben.
`time`	X	I	Gibt die Zeit, die zum Auswerten des Parameters benötigt wird, in ms zurück.
`bit_error`	I / I / R	I	Jedes der B niederwertigsten Bits von A wird mit der Wahrscheinlichkeit C verändert.
`length`	S	I	Gibt die Länge des Strings A zurück.
`substring`	S / I / I	S	Gibt den Teilstring von A zurück, der bei Position B beginnt (die erste Position ist 1) und die Länge C hat.
`ver`		S	Gibt die Versionsnummer des Interpreters zurück.
`count_bits`	I	I	Gibt die Anzahl der gesetzten Bits zurück.

Tabelle A.13: Sonstige Funktionen.

Prozedur	Parameter	Beschreibung
`mail`	X*	Gibt die Parameter in den Fenstern der anderen Anwender aus.
`writeln`	X*	Schreibt die Parameter in das eigene Anwenderfenster.
`send`	S	Schreibt den Parameter in die Eingabewarteschlange der anderen Prozesse.
`send_self`	S	Schreibt den Parameter in die Eingabewarteschlange des eigenen Prozesses.
`send_env`	S	Sendet den Parameter an die Umgebung.
`reset_mod_exp`		Löscht die Vorberechnungen für die modulare Exponentiation und gibt den entsprechenden Speicher wieder frei.
`mkdir`	S	Legt alle im Pfad angegebenen Ordner an.
`seed`	I	Initialisiert den Zufallszahlengenerator.
`void`	X	Wertet den Parameter zwar aus, nutzt ihn jedoch sonst nicht.
`store`	X / S	Speichert die angegebene Variable A in der Datei B.
`store_str`	S / S	Speichert den String A in der Datei B.
`append`	X / S	Hängt die angegebene Variable A an die Datei B an.
`append_str`	S / S	Hängt den String A an die Datei B an.
`algo`	I / I	Bestimmt den Algorithmus B für die Funktion A.
`wait`	I	Wartet die angegebene Zeit (in ms).
`lfsr_init`	I / I / I	Initialisiert das lineare zurückgekoppelte Schieberegister Nr. A mit dem Rückkopplungspolynom B und dem Initialwert C.
`lfsr_exit`	I	Gibt den Speicherplatz für Register Nr. A wieder frei.
`execute`	S	Führt das angegebene Programm aus.
`connect`	S / I	Schafft eine SDL-Verbindung zum Rechner A auf Port B.
`connect`	S / I / S	Schafft eine SDL-Verbindung zum Rechner A auf Port B mit Kanalnamen C.
`disconnect`	S / I	Beendet die SDL-Verbindung zum Rechner A auf Port B.

Literaturverzeichnis

[Dav91] Donald Watts Davies, Hrsg. *Advances in Cryptology, EUROCRYPT'91, LNCS 547*. Springer Verlag, 1991. 33

[KH92] P. Y. Kam und O. Hirota, Hrsg. *Singapore ICCS/ISITA '92 Conference Proceedings*. IEEE, 1992. 24

[N.82] N. N., Hrsg. *23th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press, 1982. 42

[OD98] Kazuo Ohta und Pei Dingyi, Hrsg. *Advances in Cryptology, ASIACRYPT'98, LNCS 1514*. Springer Verlag, 1998. 29

[TM98] Stafford E. Tavares und Henk Meijer, Hrsg. *Fifth Workshop on Selected Areas in Cryptography (SAC'98), LNCS 1556*. Springer Verlag, 1998. 35

[VPM97] Vijay Varadharajan, Josef Pieprzyk und Yi Mu, Hrsg. *Information Security and Privacy, Second Australasian Conference, ACISP'97. LNCS 1270*. Springer Verlag, 1997. 34