Entwurf und Implementierung von Plug-ins für JCrypTool

Visualisierung von kryptographischen Grundlagen

Diplomarbeit von Oryal Inel

Dezember 2009

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Theoretische Informatik
Kryptographie und Computeralgebra
Prof. Dr. Johannes Buchmann

TECHNISCHE UNIVERSITÄT DARMSTADT

Entwurf und Implementierung von Plug-ins für JCrypTool
Visualisierung von kryptographischen Grundlagen

vorgelegte Diplomarbeit von Oryal Inel

Tag der Einreichung:

Ich möchte hier die Personen erwähnen, die mich in meiner
Diplomarbeit sowie im ganzen Studium unterstützt haben.

Als erstes möchte ich mich bei meinen Eltern und meinen Pflegeeltern
für die große Unterstützung bedanken. Ohne Sie wäre das Studium nicht möglich.

Desweiteren bedanke ich mich bei meiner Freundin Miray Cetin für Ihre Geduld und Ihr
Verständnis für die Zeit, welche ich für die Anfertigung meiner Diplomarbeit brauchte.
Besonders auch für das von Ihr fleißige Korrekturlesen.

Weiter bedanke ich mich bei meinem Betreuer Dr. Vangelis Karatiolis,
der mir mit Rat und Tat beistand und sein Gehör schenkte. Er war immer ansprechbar
und half mir über die komplette Zeit, aus meiner Arbeit einen Erfolg zu machen.
Auch Prof. Dr. Johannes Buchmann möchte ich danken, dass ich in seinem Fachbereich
meine Diplomarbeit anfertigen konnte.

Zum Schluss möchte ich noch Murat Özkorkmaz danken, der mit mir zusammen Informatik
studierte, mir immer ein sehr guter Freund war und mir stets sehr gute Ratschläge gab.

Für das Korrekturlesen bedanke ich mich bei meinen Freunden
Ferit Dogan, Muhammed Oynas, Selami Kaya und Osman Kart.

Erklärung zur Diplomarbeit

Hiermit versichere ich die vorliegende Diplomarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 4. Dezember 2009

(Oryal Inel)

Konventionen

Zum besseren Verständnis gibt es einige typografische Konventionen, die in dieser Diplomarbeit befolgt werden.

- `Courier` - Codebeispiele, Klassen-, Methoden- und Dateinamen
- **Fett** - Namen der User Interface Elemente wie Menüs, Buttons, Eingabefelder, Fenstertitel und Plug-in-Name
- *kursiv* - Formeln und mathematische Konstrukte

Die Diplomarbeit wurde komplett in LaTeX geschrieben. Für die äußere Gestaltung wurde das Corporate Design der **Technischen Universität Darmstadt** gewählt. Für den internen Aufbau der Diplomarbeit diente u.a. folgende Literatur: [MG00], [Kop00] und [Kna04]. Die Zustandsdiagramme wurden mit dem Programm **ArgoUML**[1] erstellt. Für den Entwurf von Eclipse Plug-ins, dienten die Bücher [EC09] und [DG03].

[1] http://argouml.tigris.org

1 Einleitung

Diese Diplomarbeit beschreibt die Implementierung von drei Plug-ins, für die kryptographische E-Learning-Plattform **JCrypTool**[1]. **JCrypTool** ist ein als Eclipse Rich Client Plattform (RCP) entwickeltes Open Source Projekt, die es Studenten, Lehrer und anderen an der kryptographie Interessierten ermöglicht, einen einfachen und modernen Einblick in die kryptographischen Verfahren zu erhalten. Das Ziel ist es, dabei eine neue Form von E-Learning zu schaffen, indem die Benutzer nicht nur über die Kryptographie lernen, sondern auch die Entwicklung ihrer eigenen kryptographischen Plug-ins, um somit die **JCrypTool** Plattform auf unterschiedliche Weise zu erweitern.

In dieser Einführung gehen wir kurz auf die RCP-Plattform ein und erläutern wie man ein Plug-in erstellen kann. In den Kapiteln 2 – 4 stellen wir die drei entwickelten Plug-ins vor und schliessen die Diplomarbeit mit dem Kapitel 5 **Ausblick** ab.

1.1 Die Eclipse Rich Client Plattform

Eclipse[2] ist eine moderne integrierte Entwicklungsumgebung (IDE) für die Softwareentwicklung. Ursprünglich wurde sie von IBM[3] für die Programmiersprache Java entwickelt und ist seit 2001 als Open Source Framework freigegeben. Zur Zeit liegt Eclipse im Galileo Release in der Version 3.5 vor. Aufgrund seiner Erweiterbarkeit gibt es Plug-ins für fast jede Programmiersprache, dabei werden nicht nur Programmiersprachen unterstützt, sondern auch Modellierungssprachen, wie z.B. UML[4]. Mit der Rich Client Plattform bietet Eclipse Anwendungsentwickler eine Möglichkeit, basierend auf dem Eclipse Framework von der Eclipse-IDE unabhängige Anwendungen zu schreiben. Dabei besteht eine RCP Anwendung aus mindestens der Eclipse Core Platform, die den Lebenszyklus einer Eclipse Anwendung steuert, dem Standard Widget Toolkit (SWT), eine Bibliothek, die graphische Oberflächen erstellt und JFace, die aus den von SWT gelieferten Basiskomponenten komplexere Widgets zusammenstellt.

1.2 Erstellung eines Eclipse Plug-ins

Um ein Plug-in zu erstellen, kann man den Eclipse Wizard für **Plug-in Project** verwenden. Es werden nun verschiedene Templates für die verschiedenen Plug-ins angezeigt, siehe Abbildung 1.1.

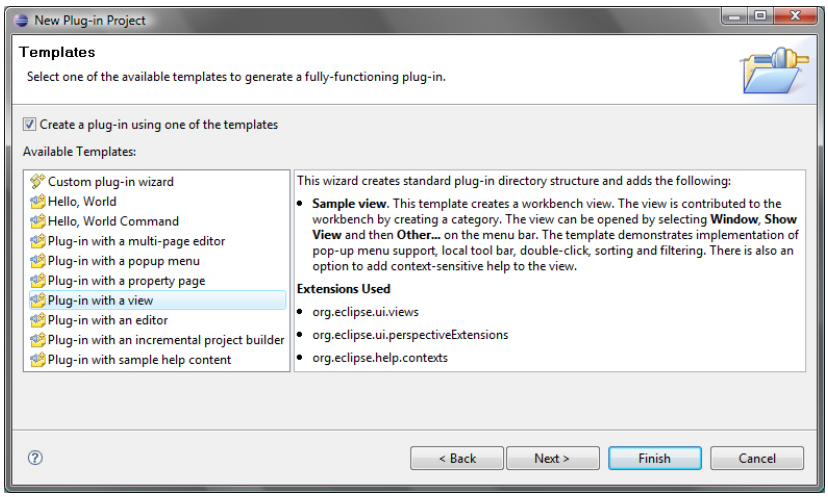

Abbildung 1.1: Der Eclipse Wizard für die verschiedenen Plug-in Formen

[1] http://jcryptool.sourceforge.net/JCrypTool/Home.html
[2] http://www.eclipse.org
[3] http://www.ibm.com
[4] **U**nified **M**odeling **L**anguage: http://www.uml.org/

Da unsere entwickelten Plug-ins Implementierungen einer **View** sind, wählen wir das Template **Plug-in with a view** aus und bestätigen mit dem **Next** Button. In dem nächsten Dialogfeld kann der Klassenname und die Kategorie der View eingetragen werden, in der die View des Plug-ins aufgelistet werden soll. Die Kategorie wird dann später durch die **JCrypTool** Kategorie `org.jcryptool.visual` ersetzt. Es werden die nötigen Dateien von Eclipse für eine Beispielview generiert und man kann beginnen seine eigene **View** zu implementieren. Jedes Plug-in hat eine `plugin.xml` Datei, in der man Einstellungen zum Plug-in selbst vornehmen kann. Dabei können Einstellungen zu ID, Name und Herausgeber des Plug-ins, sowie auch die **Dependencies** zu anderen Plug-in und auch die **Extensions**, die das Plug-in besitzen soll, festgelegt werden.

1.3 Das JCrypTool Projekt

Das **JCrypTool** Projekt kann man unter der Adresse `http://jcryptool.sourceforge.net/JCrypTool/Home.html` finden. Das Projekt unterteilt sich einmal in das **JCrypTool Core Project**[5] und in das **JCrypTool Plug-ins Project**[6]. Das Core Projekt beinhaltet die Grundkomponente der Anwendung und ist für die Ausführung der Anwendung notwendig. Das JCrypTool Plug-in Projekt ist eine Sammlung von Plug-ins für die **JCrypTool** Plattform. Der schematische Aufbau der einzelnen Komponenten des **JCrypTool Core Projects** sind in Abbildung 1.2 und die Komponenten des **JCrypTool Plug-in Projects** sind in Abbildung 1.3 dargestellt. Die hier in der Diplomarbeit vorgestellten Plug-ins sind Visualisierungen von Algorithmen. Somit werden sie in **JCrypTool** in der Rubrik **Visuals** aufgelistet. Dafür fügen wir die entwickelten Plug-ins in das Feature Projekt `org.jcryptool.visual.feature` hinzu. Ein Feature-Projekt ist eine Liste von Plug-ins oder auch anderen Features, um Plug-ins zu gruppieren und die Übersichtlichkeit zu erhöhen.

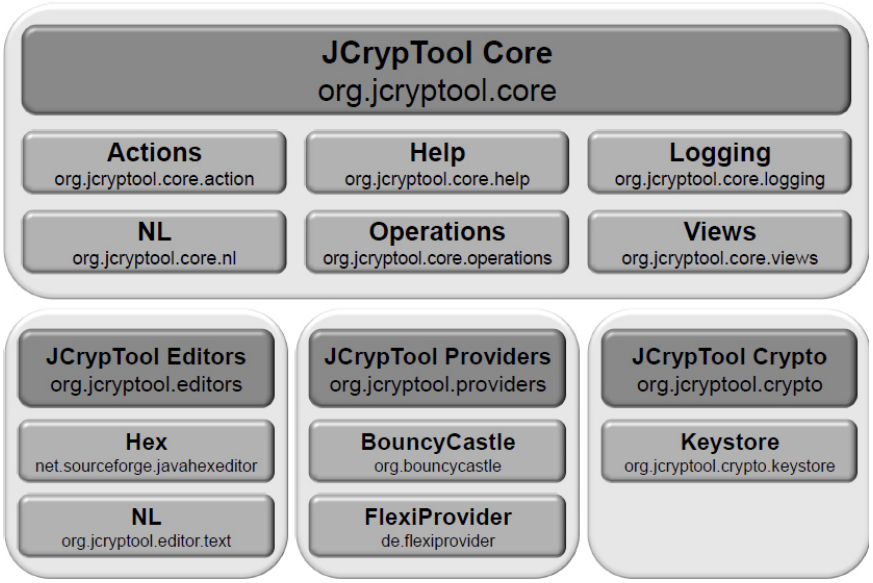

Abbildung 1.2: Die einzelnen Komponenten des JCrypTool Core Projects, entnommen aus [JCt09]

[5] http://sourceforge.net/projects/jcryptool/
[6] http://sourceforge.net/projects/jctplugins/

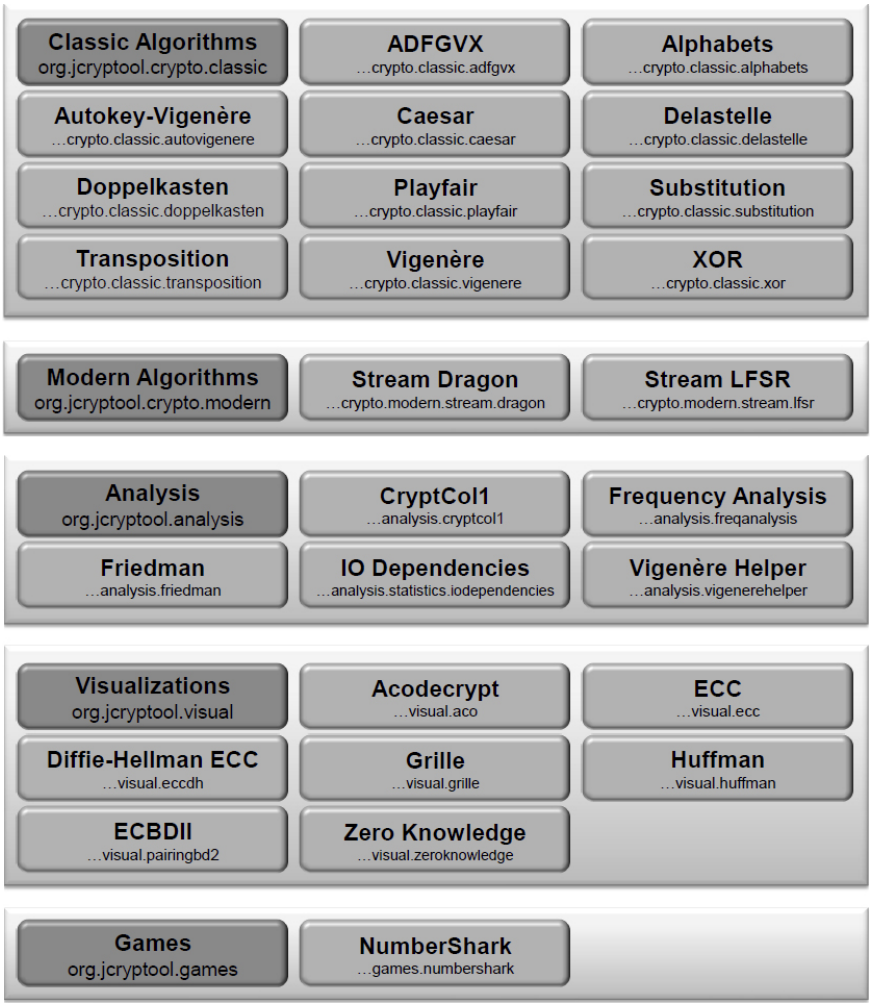

Abbildung 1.3: Die einzelnen Komponenten des JCrypTool Plug-ins Projects, entnommen aus [JCt09]

1.3.1 Die Kontext-Hilfe

Jedes Plug-in besitzt eine Kontext-Hilfe, die man im Menü aufrufen kann. Damit ein Plug-in durch eine Hilfefunktion unterstützt wird, muss man in der `plugin.xml` Datei die **Extention** `org.eclipse.help.toc` hinzufügen. Dieser Extention kann man nun eine **Table of Content** XML-Datei (toc.xml) hinzufügen, die das Inhaltsverzeichnis der Hilfe enthält. In dieser Datei kann man die Struktur der Hilfe festlegen, dabei wird der Entwickler durch Eclipse beim Aufbau des Inhaltsverzeichnisses unterstützt. Er kann durch Klicken auf die rechte Maustaste die Elemente **Topic**, **Link** und **Anchor** auswählen und die Pfade zu den Hilfedateien festlegen. Wichtig hierbei ist, dass man in den Einstellungen der `toc.xml` Datei den **Anchor** auf `/org.jcryptool.core.help/$nl$/toc_users_guide.xml#visualizations` setzt. Dies bewirkt, dass die Hilfe des Plug-ins in der **JCrypTool** Kontext-Hilfe im Anwendungshandbuch unter Visualisierungen erreichbar ist. Nun kann man die Hilfe mit beliebigen HTML-Text beschreiben.

1.3.2 Internationalisierung

Die hier vorgestellten Plug-ins unterstützen mehrere Sprachversionen. Dabei kann man sowohl die User Elemente als auch die Kontext-Hilfe in verschiedenen Sprachen implementieren. Eclipse bietet dabei einen Mechanismus, um die Namen der User-Elemente zu extrahieren. Die hier entwickelten Plug-ins werden sowohl in deutscher als auch in englischer Sprache umgesetzt. Man kann die Sprache über die Einstellungen von **JCrypTool** ändern. Um die verschiedenen Sprachversionen bei der Entwicklung zu testen, genügt es bei den Programmparameter den Wert `-nl $(target.nl)` durch den jeweiligen Ländercode zu ersetzten, für die deutsche Sprache ist das `-nl de`.

Um die Namen der User-Elemente zu extrahieren, benutzen wir den **Externalize Strings** Mechanismus von Eclipse. Dabei werden alle Strings in einer Klasse mit einem **Key** versehen. In einem Dialogfeld kann man nun die Strings, die nicht extrahiert werden sollen abwählen und den Strings, die extrahiert werden sollen, einen aussagekräftigen Namen geben. Eclipse generiert nun die Dateien `Messages.java` und `messages.properties`. In der `messages.properties` Datei stehen nun alle extrahierten Strings und die dazugehörigen Keys, die eine Hashtabelle repräsentieren. Die Datei `Messages.java` dient dazu den jeweiligen String in dem Quellcode zur Laufzeit zu ersetzen. Im Quellcode wird nun der extrahierte String mittels des Aufrufs `Messages.keyID` durchgeführt. Dabei ist `keyID` der Key aus der Datei `messages.properties`. Um nun eine weitere Sprache anzubieten, erzeugt man eine neue Datei Namens `messages_<Ländercode>.properties` und ersetzt in dieser den Bereich `<Ländercode>` durch den jeweiligen Ländercode der Sprache. Eine Liste von ISO Sprachcodes kann man auf den Seiten des **Unicode Consortium**[7] finden. Für eine deutsche Sprachversion erzeugt man nun eine Datei `messages_de.properties`. Nun kann man den Inhalt der orginal `messages.properties` Datei kopieren und in die neu erzeugte Datei einfügen. In der Datei werden nun die originalen Bezeichnungen übersetzt und abgespeichert. Für weitere Sprachversionen erzeugt man weitere `messages_<Ländercode>.properties` Dateien und geht wie beschrieben vor.

Für eine andere Sprachversion der Kontext-Hilfe erzeugt man ein Verzeichnis `nl`, dabei steht die Abkürzung nl für national language. Dieses Verzeichnis enthält Unterverzeichnisse mit den Namen der Ländercodes, für die verschiedenen Sprachversionen. Darin können nun die Hilfedateien in einem Ordner `help` abgelegt werden. Die Abbildung 1.4 zeigt die Verzeichnisstruktur der deutschen Hilfe für das **Extended Euclidean Plug-in**.

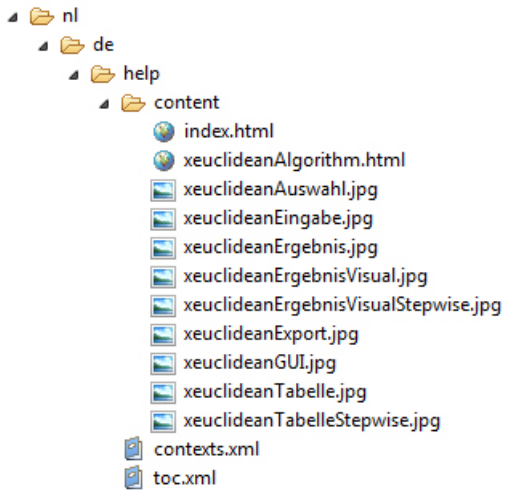

Abbildung 1.4: Verzeichnisstruktur der Kontext-Hilfe

1.4 Ziele und Motivation

Das Ziel dieser Diplomarbeit ist es, die Algorithmen **Extended Euclidean**, **Chinese Remainder Theorem** und **Shamir's Secret Sharing** aus dem Buch [Buc08] als Plug-in für die E-Learning Plattform **JCrypTool** zu implementieren. Dabei liegt die Motivation darin begründet, die Algorithmen für den Benutzer leicht zugänglich zu gestalten und dadurch das Verständnis über den Ablauf der Algorithmen dem Benutzer näher zu bringen. Es wird großen Wert auf die Usability der Plug-ins gelegt. Unter Usability verstehen wir, dass die Struktur der Algorithmen an sich erkennbar ist. Um dies zu erreichen, sollen gewisse Berechnungsschritte der Algorithmen farbig hervorgehoben werden, um somit die Vorgehensweise des Algorithmus zu verstärken. Der Ablauf für die jeweiligen Algorithmen soll so erklärt werden, dass der Benutzer nach Durchlauf des Plug-ins den Algorithmus versteht und selbstständig anwenden kann. Dabei ist es wichtig, den Benutzer schrittweise an den Algorithmus heranzuführen und nicht nur das Endergebnis anzuzeigen. Ziel ist es, die Plug-ins so zu entwerfen, dass sie sowohl für Lehrzwecke als auch für reale Anwendungen einsetzbar sind. Die E-Learning Plattform **JCrypTool** wird verstärkt in einigen Schulen und Hochschulen eingesetzt, deshalb ist es wichtig, die Berechnung der Algorithmen in eine ausdruckbare Form zu bringen, um den Lerneffekt zu unterstützen. Auch soll die Theorie der Algorithmen in den Plug-ins enthalten und abrufbar sein.

[7] http://unicode.org/repos/cldr-tmp/trunk/diff/supplemental/languages_and_scripts.html

2 Das Extended Euclidean Plug-in

Der **Extended Euclidean** Algorithmus ist einer der grundlegenden Algorithmen in der Kryptographie. Mit dem **Extended Euclidean** läßt sich der größte gemeinsame Teiler von zwei Zahlen berechnen. Der Algorithmus wird u.a. für das RSA[1] Kryptosystem zur Schlüsselerzeugung verwendet, beschrieben in [RSA78]. Desweiteren wird der Algorithmus im **Chinese Remainder Theroem** verwendet, der in Kaptiel 3 beschrieben wird. Weitere Anwendungsgebiete des **Extended Euclidean** Algorithmus sind, z.B. Primzahlgenerator und Primzahltests.

Die Idee für die Entwicklung des Plug-ins ist es, den Algorithmus für den Benutzer leicht verständlich zu machen. Durch die Benutzung des Plug-ins, soll dem Benutzer die Funktionsweise des Algorithmus näher gebracht werden. Das Plug-in kann sowohl von Studierenden, zum Erlernen des Algorithmus benutzt werden, als auch vom Lehrkörper, um z.B. Übungs- oder Klausuraufgaben zu generieren.

In Abschnitt 2.1 wird der **Extended Euclidean** Algorithmus mathematisch erklärt. Der Entwurf des Plug-ins wird in Abschnitt 2.2 beschrieben, die Implementierung in Abschnitt 2.3 dargelegt und in Abschnitt 2.4 auf die Funktionsweise des Plug-ins eingegangen.

2.1 Der Extended Euclidean Algorithmus

Der **Extended Euclidean** Algorithmus berechnet den größten gemeinsamen Teiler zweier ganzen Zahlen p und q. Zusätzlich liefert der Algorithmus zwei ganze Zahlen x, y, so daß $gcd(p,q) = px + qy$ gilt. Der Algorithmus wurde aus dem Buch [Buc08] entnommen und für `BigInteger` angepasst. Den Beweis, dass es genau einen größten gemeinsamen Teiler zweier ganzer Zahlen p, q gibt, kann man in [Buc08] nachlesen. Der Vollständigkeitshalber wird der größte gemeinsame Teiler von 0 und 0 auf 0 gesetzt, also ist $gcd(0,0) = 0$. Somit ist der größte gemeinsame Teiler zweier ganzer Zahlen nie negativ. Wir übernehmen aus [Buc08] folgendes Theorem.

Theorem 2.1.1 *1. Wenn $q = 0$ ist, dann ist der $gcd(p,q) = |p|$.*
2. Wenn $b \neq 0$ ist, dann ist $gcd(p,q) = gcd(|q|, p \ mod \ |q|)$.

Der Algorithmus wird anhand eines Beispiels erläutert.

Beispiel Wir möchten den $gcd(123,23)$ berechnen. Nach Theorem 2.1.1 erhalten wir $gcd(123,23)$ $= gcd(23, 123 \ mod \ 23) = gcd(23,8) = gcd(8,7) = gcd(7,1) = gcd(1,0) = 1$.

Zuerst wird überprüft welche der beiden Zahlen größer ist. Die größere Zahl wird als Parameter eins, die kleinere als Parameter zwei deklariert. Falls beide Zahlen identisch sind, werden keine Änderung vorgenommen, da in diesem Fall der $gcd(p,p) = p$ schon eindeutig bestimmt ist. Die Funktion wird mit $gcd(p,q)$ aufgerufen. Nun wird solange die Zahl p durch die Zahl q geteilt und modulo gerechnet, bis die Zahl q Null wird. Für die nächste Iteration wird q das neue p und das neue q ist der Rest der Moduloberechnung $q = p \ mod \ q$. Da q immer der Rest der Moduloberechnung ist und dieser irgendwann Null sein wird, wird sichergestellt, dass der Algorithmus terminiert. Man kann die Berechnung kompakt in einer Tabelle aufschreiben, siehe Tabelle 2.1.

Index	Quotient	Remainder
0		123
1	5	23
2	2	8
3	1	7
4	7	1
5		0

Tabelle 2.1: Tabellenschreibweise

[1] RSA ist ein asymmetrisches Kryptosystem, das sowohl zur Verschlüsselung als auch zur digitalen Signatur verwendet werden kann

Um die Werte für X und Y zu berechnen, initialisieren wir die ersten zwei Iterationen folgendermaßen:

$$x_0 = 0, \qquad x_1 = 1 \qquad und \qquad y_0 = 1, \qquad y_1 = 0. \tag{2.1}$$

Mit $r_0, \ldots, r_{n+1}$ bezeichen wir die Remainderfolge und mit $q_1, \ldots, q_n$ die Folge der Quotienten, die bei der Anwendung des **Extended Euclidean** Algorithmus auf p, q entstehen. Die Konstruktionsweise der zwei Folgen (x_k) und (y_k) mit der Eigenschaft $x = (-1)^n x_n$ und $y = (-1)^{n+1} y_n$ wird durch folgende Formeln beschrieben.

$$x_{k+1} = q_k x_k + x_{k-1}, \qquad y_{k+1} = q_k y_k + y_{k-1}, \qquad 1 \leq k \leq n. \tag{2.2}$$

Dies läßt sich wiederum für unser Ausgangsbeispiel $gcd(123, 23)$ in eine Tabelle überführen, siehe Tabelle 2.2.

Index	Quotient	Remainder	X	Y
0		123	1	0
1	5	23	0	1
2	2	8	1	5
3	1	7	2	11
4	7	1	3	16
5		0	23	123

Tabelle 2.2: Tabellenschreibweise mit Koeffizienten X und Y

Somit kann man den größten gemeinsamen Teiler von p und q als Linearkombination von p und q darstellen.

$$gcd(123, 23) = 1 = 123 * 3 + 23 * (-16) \tag{2.3}$$

Der **Extended Euclidean** liefert uns neben dem größten gemeinsamen Teiler $gcd(p, q)$ auch die Koeffizienten

$$x = (-1)^n x_n \qquad y = (-1)^{n+1} y_n \tag{2.4}$$

2.2 Der Entwurf des Plug-ins

Das Plug-in ist als **View** implementiert. Dabei wird das Eclipse Framework benutzt. Die **View** erbt, wie alle anderen **Views** in **JCrypTool**, von der abstrakten Klasse `ViewPart`, die ein Teil des Eclipse Frameworks ist.

Das **Extended Euclidean** Plug-in gliedert sich in 5 Packages:

- `org.jcryptool.visual.xeuclidean`
- `org.jcryptool.visual.xeuclidean.algorithm`
- `org.jcryptool.visual.xeuclidean.export`
- `org.jcryptool.visual.xeuclidean.test`
- `org.jcryptool.visual.xeuclidean.views`

Das **Extended Euclidean** Plug-in ist in Abblidung 2.1 schematisch dargestellt. Die einzelnen Elemente des User Interfaces werden in Tabelle 2.3 beschrieben. Es beschreibt den initialen Ausgangszustand des Plug-ins, wenn die **View** zum ersten mal aufgerufen wird. Dabei gibt die Spalte **Typ** das `Widget` Objekt des **SWT**s an, dass bei der Implementierung benutzt wurde. Der **initiale Zustand** gibt an, ob das `Widget` initial aktiviert oder deaktiviert ist. Zum Beispiel sind die Exportfunktionen und einige Buttons zu Beginn deaktiviert und werden erst nach Ablauf einer Berechnung aktiviert. Die Spalte **Text** gibt den verwendeten Text des `Widgets` an, wobei die Eingabefelder leer sind. Aus der letzte Spalte kann noch eine kurze Beschreibung zum verwendeten `Widget` entnommen werden.

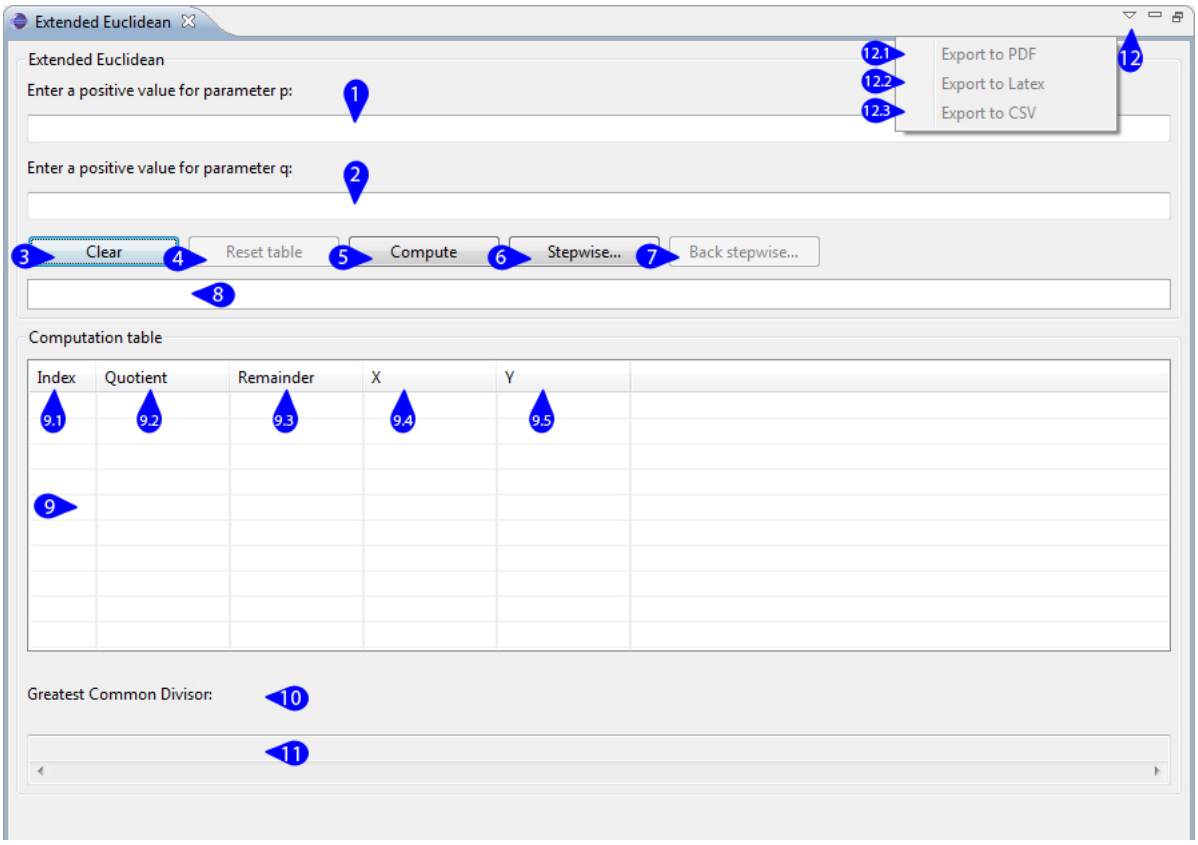

Abbildung 2.1: Das User Interface des Extended Euclidean Plug-ins

#	Typ	Initialer Zustand	Text	Beschreibung
1	Textfield	aktiviert	leer	Eingabefeld des Parameters **p** als `BigInteger` Zahl.
2	Textfield	aktiviert	leer	Eingabefeld des Parameters **q** als `BigInteger` Zahl.
3	Button	aktiviert	Clear	Löscht alle Eingabefelder und die Ausgabetabelle. Es wird der initiale Ausgangszustand wiederhergestellt.
4	Button	deaktiviert	Reset table	Löscht die Ausgabetabelle. Der Button wird erst aktiviert, falls eine Berechnung durchgeführt wurde.
5	Button	aktiviert	Compute	Führt die Berechnung des **Extended Euclidean** aus.
6	Button	aktiviert	Stepwise ...	Führt die Berechnung des **Extended Euclidean** schrittweise aus.
7	Button	deaktiviert	Back stepwise ...	Geht in der Berechnung einen Schritt zurück. Der Button wird erst aktiviert sobald Button **5** oder **6** ausgeführt wurde.
8	Textfield	deaktiviert	leer	Zeigt Informationen über den aktuellen Zustand der Berechnung an.
9	Table	deaktiviert	leer	In der Tabelle wird die Berechnung des **Extended Euclidean** schrittweise angezeigt.
9.1	Table Column	deaktiviert	Index	Die Spalte **Index**, gibt die aktuelle Position an, in der man sich in der Berechnung befindet.
9.2	Table Column	deaktiviert	Quotient	In dieser Spalte wird der aktuelle **Quotient** der Berechnung angezeigt.
9.3	Table Column	deaktiviert	Remainder	In dieser Spalte wird der **Remainder** der aktuellen Berechnung angezeigt.
9.4	Table Column	deaktiviert	X	Hier wird das jeweilige **X** angezeigt.
9.5	Table Column	deaktiviert	Y	Hier wird das jeweilige **Y** angezeigt.
10	Label	deaktiviert	leer	Darstellung der Berechnungsformel für den **Extended Euclidean**.
11	Textfield	deaktiviert	leer	In diesem Textfeld wird das Endergebnis mit den Parametern angezeigt.
12	Menu	deaktiviert	$\bigtriangledown$	Hier kann nach Durchführung der Berechnung das Ergebnis als **PDF**, **CSV** oder **LaTeX** Dateiformat exportiert werden. Das Menü wird erst nach Durchführen einer Berechnung aktiv.
12.1	Menu Item	deaktiviert	Export to PDF	Ergebnis der Berechnung als **PDF** Datei exportieren.
12.2	Menu Item	deaktiviert	Export to CSV	Ergebnis der Berechnung als **CSV** Datei exportieren.
12.3	Menu Item	deaktiviert	Export to LaTeX	Ergebnis der Berechnung als **LaTeX** Datei exportieren.

Tabelle 2.3: Die Verhaltensbeschreibung des Extended Euclidean Plug-ins

Das Plug-in hat zum Einen den **Compute** Modus, in dem man eine Berechung vollständig ablaufen lassen kann und zum Anderen den **Stepwise** Modus, in dem man sich alle Einzelschritte anzeigen lassen kann. Desweiteren gibt es die Möglichkeit, sich die Endergebnisse expotieren zu lassen. Für den schematischen Ablauf wurde ein UML-Aktivitätsdiagramm gewählt, wie in [CR07] beschrieben, um die einzelnen Schritte besser darzustellen, siehe Abbildung 2.2. Im folgenden soll auf die Implementierung des Plug-ins eingegangen werden.

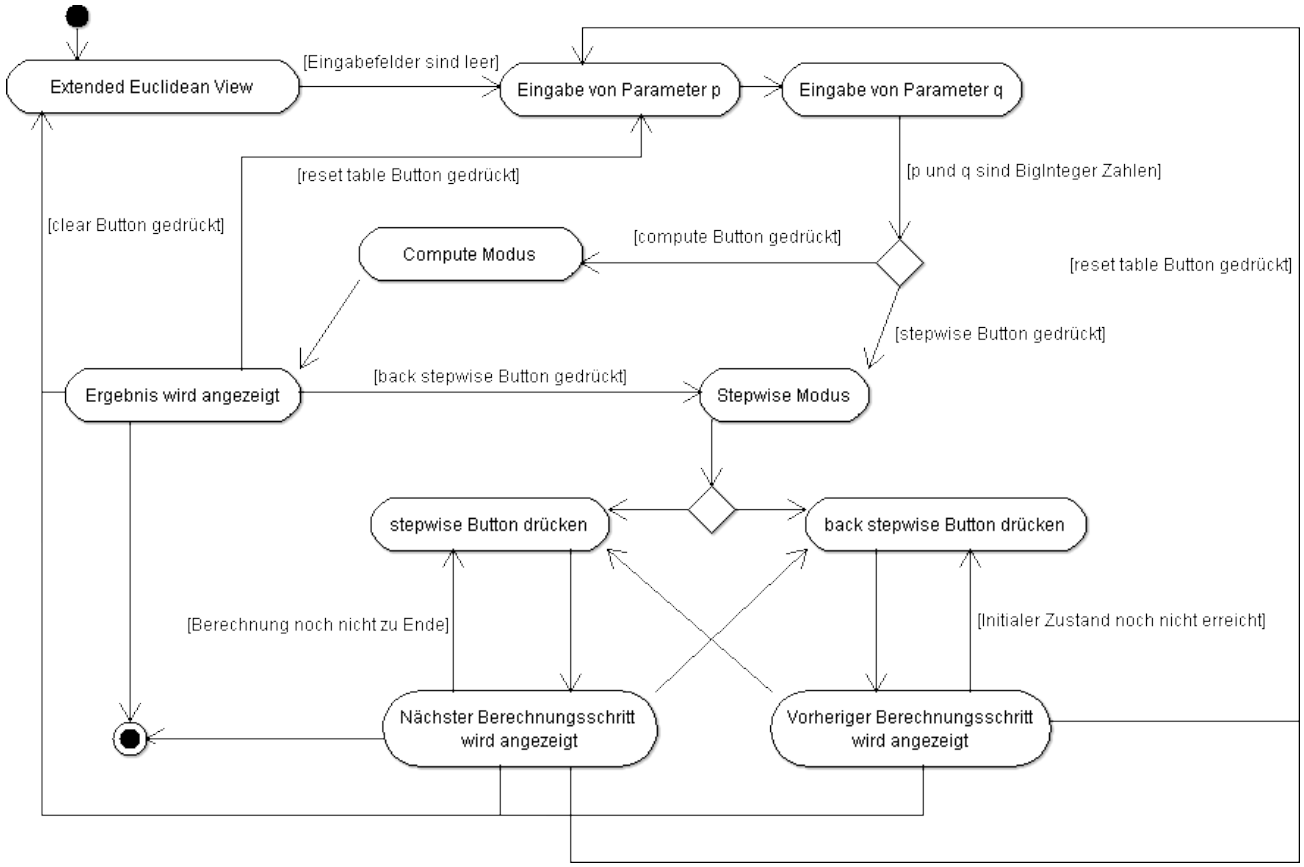

Abbildung 2.2: Das UML Aktivitätsdiagramm des Extended Euclidean Plug-in

2.3 Die Implementierung

In der Klasse `XEuclideanView` ist die Darstellung der **View** umgesetzt. Es wurde darauf geachtet, die Funktionalität und die Darstellung des Plug-ins zu kapseln. Somit ist die Funktionalität des Algorithmus und die Darstellung der **View** jeweils in seperate `packages` aufgeteilt worden. Dies enspricht dem **Model View Controller** Entwurfsmuster, das in [EG09] beschrieben wird. In diesem Plug-in entspricht das **Model** der Klasse `XEuclid` und die **View** und der **Controller** sind zusammengefaßt in der Klasse `XEuclideanView`. Das Plug-in besteht aus zwei Eingabefeldern, mehreren Buttons, Ausgabefeldern und einer Tabelle. Für die Aufnahme der zwei Parameter wurden `org.eclipse.swt.widget.Text` Eingabefelder verwendet. Zusätzlich wurde jedem Eingabefeld ein `VerifyListener` hinzugefügt, der dafür sorgt, dass nur verifizierte Parameter eingegeben werden können. In unserem Fall sind dies positive Zahlen ohne führende Nullen. Der `VerifyListener` ist ein `Interface`, das nur die Methode `verifyText(VerifyEvent e)` beinhaltet. Sobald per Tastatureingabe der Text im Textfeld modifiziert wird, wird die Methode `verifyText(VerifyEvent e)` aufgerufen, die überprüft, ob es sich um einen gültigen Wert handelt. Der dazu gehörige Quellcode für den Parameter *p* ist in Listing 2.1 abgebildet.

```
1  private Text pText = new Text(action, SWT.BORDER);
2  pTextOnlyNumbers = new VerifyListener() {
3     public void verifyText(VerifyEvent e) {
4        /** keyCode == 8 is BACKSPACE and keyCode == 48 and keyCode == 128 is DEL */
5        if (e.text.matches("[0-9]") || e.keyCode == 8 || e.keyCode == 127){
6
7           if (pText.getText().length() == 0 && e.text.compareTo("0") == 0){
8              e.doit = false;
9           } else if(pText.getSelection().x == 0 && e.keyCode == 48){
10             e.doit = false;
11          }
```

```
12              else {
13                  e.doit = true;
14              }
15          }else{
16              e.doit = false;
17          }
18      }
19  };
20  pText.addVerifyListener(pTextOnlyNumbers);
```

Listing 2.1: VerifyListener für das Textfield

Zuerst wird auf korrekte Eingabe geprüft. Dies wird mit der Methode `e.text.matches("[0-9]")` durchgeführt, die einen regulärern Ausdruck verwendet, um zu überprüfen, ob es sich um eine Zahl handelt. Zusätzlich wird noch die **Backspace** und die **Entfernen** Taste zugelassen, um eine eventuelle Fehleingabe des Benutzers zu korrigieren. Falls die Eingabe nicht dem regulären Ausdruck entspricht, wird die Eingabe abgelehnt. Dies wird mit Setzen der Variable `e.doit` auf `false` durchgeführt. Durch den regulären Ausdruck verbieten wir sowohl Groß- als auch Kleinbuchstaben, Sonderzeichen und Funktionstasten. Führende Nullen sind zwar keine Fehleingabe, werden aber ebenfalls bei der Überprüfung abgefangen. Dies erhöht die Usability des Plug-ins. Dazu wird mit der Methode `pText.getText().length() == 0` die Länge der Eingabe, die schon in dem Eingabefeld steht, abgefragt. Falls dies die Länge Null hat, wissen wir, dass das Eingabefeld noch leer ist. Zusätzlich überprüfen wir, ob das eingegebene Zeichen eine Null ist, um die besagten führenden Nullen nicht zuzulassen. In diesem Fall wird die Funktion mit `e.doit = false` verlassen. Im `else if` Teil wird noch der Fall abgefangen, in dem im Eingabefeld schon ein gültiger Wert drin steht und der Benutzer mit den Cursor Tasten zum Anfang der Eingabe geht, um dort eine Eingabe einzufügen. Zum Beispiel könnte man vor eine Zahl 80, eine führende Null schreiben. Dies wird mit der Methode `pText.getSelection().x` abgefragt, wobei `x` die Startposition des Cursors ist. Falls der Rückgabewert von `x` gleich Null ist und der `keyCode == 48`, dann wissen wir, dass sich der Cursor am Anfang befindet und eine Null eingegeben wurde. Für alle anderen Fälle wissen wir, dass die Eingabe des Benutzers korrekt ist und die Werte werden in das Textfeld übernommen.

2.3.1 Der Stepwise Modus

Der **Stepwise** Modus wurde als **endlicher Automat**[2] implementiert. Wir speichern uns den Zustand in einer Variable. Seit der Version 5 unterstützt Java[3] das Enumeration-Konzept, dass wir hier benutzen und sich für die Modellierung der Zustände sehr gut eignet. Vorteil der Enumeration ist, dass es zum Einen typsicher ist und zum Anderen man über ein `switch` Statement die Zustände iterieren kann. Unser **endlicher Automat** hat sechs Zustände, dargestellt in Listing 2.2.

```
1  private enum visualMode { INIT, QUOTIENT, REMAINDER, X_EVAL, Y_EVAL, RESULT }
2
3  private visualMode nextState;
4  nextState = visualMode.INIT;
5
6  private visualMode previousState;
7  previousState = null;
```

Listing 2.2: Endlicher Automat mittels Enumeration

In der Variable `nextState` speichern wir den nächsten Zustand, der beim Start mit dem Zustand `INIT` initialisiert wird. Die Variable `previousState` dient dazu, sich den vorherigen Zustand zu merken, um in der Berechnung des Algorithmus einen Schritt zurück zugehen. Zusätzlich zu unserem endlichen Automaten haben wir noch eine Variable `stepwiseCounter`, in der wir die aktuelle Tabellenzeile merken.

Durch Klicken auf den Button **Stepwise...** wird die Methode `mouseUp(final MouseEvent e)` ausgeführt. Die Methode gehört zu der Klasse `MouseAdapter`, die für die Mausklick-Events zuständig ist. In dieser Methode definieren wir die Funktionalität unseres endlichen Automaten. Der endliche Automat wird durch ein `switch` Statement beschrieben, der über die Variable `nextState` iteriert. Im `INIT` Zustand wird zuerst die Methode `clear()` aufgerufen, die die Löschung

[2] im englischen **F**inite **S**tate **M**achine (FSM)

[3] http://www.java.com

der Tabelle vornimmt und den Tabellencounter zurücksetzt. Falls die Eingabefelder nicht leer sind, wird der **Extended Euclidean** Algorithmus mit den Werten `p` und `q` ausgeführt. Die `XEuclid` Klasse bietet mehrere Getter Methoden, um die Daten der Berechnung zur Verfügung zu stellen. Die Folgen für *Quotient, Remainder, X* und *Y* werden in `Vectoren` vom Typ `BigInteger` gespeichert. Nun werden die ersten beiden Zeilen der Tabelle ausgefüllt, dass der Initialisierung des Algorithmus entspricht. Desweiteren wird im Visualisierungsfeld die Nachricht **Initialization** angezeigt. In diesem Feld werden Informationen über die aktuelle Berechnung angezeigt. Um das Visualisierungsfeld farbig hervorzuheben, wurde die Klasse `StyledText` verwendet. Dort ist es möglich, den Text zu färben oder auch andere Textformatierungen zu benutzen. Zum Schluß wird der `nextState` auf den Zustand `QUOTIENT` gesetzt und `previousState = null` zugewiesen.

Im nächsten Zustand `QUOTIENT` wird der Button **Back stepwise...** aktiviert, denn ab diesem Zeitpunkt kann man einen Berechnungsschritt zurückgehen. Wenn man in dem Zustand `QUOTIENT` ist und den Button **Back stepwise...** drückt, gelangt man zu dem Zustand `Y_EVAL`. Nur für den Fall, dass es keinen vorherigen `Y_EVAL` Zustand gibt, ist man wieder im initialen Zustand `INIT`. Um dies zu garantieren, wird die Variable `previousState` abgefragt. Falls diese Variable ungleich `null` ist, wird der `previousState` auf `Y_EVAL` gesetzt. Andernfalls befinden wir uns beim Drücken des **Back stepwise...** Buttons im `INIT` Zustand. Desweiteren werden die Parameter für die Berechnung des Quotienten mit **rot** und **grün** eingefärbt und in die Tabelle eingetragen. Das Ergebnis der Berechnung wird mit der Farbe **blau** dargestellt. In dem Visualisierungsfeld wird die Berechnungsvorschrift angezeigt und farbig hervorgehoben. Dabei stimmen die Farben der Tabelle mit denen des Visualisierungsfeldes überein. Durch Betätigen des **Stepwise...** Buttons gelangen wir in den nächsten Zustand: `REMAINDER`.

Dieser Zustand gibt den Rest der Moduloberechnung aus und färbt sie **blau**, da es sich um ein Endergebnis handelt. Der Wert des Quotienten wird in **schwarz** gefärbt, da es für die aktuelle Berechnung nicht nötig ist, den Quotienten hervorzuheben. Desweiteren wird wiederum im Visualisierungfeld die Information der Berechnung angezeigt. Es werden die Variablen `nextState` auf `X_EVAL` und `previousState` auf `QUOTIENT` gesetzt und auf die nächste Benutzereingabe gewartet.

Die Zustände `X_EVAL` und `Y_EVAL` sind für die Darstellung der Berechnug der Koeffizienten *X* und *Y* zuständig. Das Verhalten beider Zustände ist ähnlich. Sie schreiben in die Tabelle und in das Visualisierungsfeld die Ergebnisse der Berechnung und heben die Werte farbig hervor. Da wir diesmal drei Paramter haben, wird der Quotient mit der Farbe **violett** dargestellt und das Ergebnis wiederum mit **blau**. Der Unterschied beider Zustände, abgesehen davon, dass wir sie in verschiedenen Tabellenspalten schreiben, ist der Wechsel in den nächsten bzw. vorherigen Zustand. Von Zustand `X_EVAL` wird in den Zustand `Y_EVAL` gewechselt. Wenn man in Zustand `Y_EVAL` ist, gibt es zwei Zustände, in die man gelangen kann, je nachdem in welchem Berechnungsschritt man sich befindet. Falls wir im letzten Berechnungschritt sind, müssen wir in den letzten Zustand `RESULT` wechseln. Dies überprüfen wir, indem wir den Wert des `stepwiseCounters` mit der Anzahl der Werte in der Quotientspalte vergleichen. Falls dieser Wert größer ist, wissen wir, dass die Berechnung im letzten Schritt ist. Andernfalls müssen wir in den Zustand `QUOTIENT` wechseln, den `stepwiseCounter` erhöhen und mit der nächsten Iteration der Berechnung fortfahren. Unabhängig davon in welchen Zustand wir wechseln, muss der `previousState` auf `X_EVAL` gesetzt werden.

Der Endzustand unseres **endlichen Automaten** ist der `RESULT`-Zustand. Dieser dient lediglich dazu, die Endergebnisse anzuzeigen und die Buttons **Stepwise...** und **Compute** zu deaktivieren. Unten werden in dem Berechungsfeld die Formel und das Ergebnis samt Parameter in einem Textfeld dargestellt, um das Herauskopieren der Werte zu erleichtern. Der `nextState` wird auf `null` gesetzt, da wir uns im Endzustand befinden. Der `previousState` wird auf den Zustand `Y_EVAL` gesetzt. Zusätzlich werden die Exportfunktionen im Menü aktiviert. Zu beachten ist, dass der **Stepwise** Modus von jedem Zustand aus durch Betätigen des **Compute** Buttons unterbrochen und in den **Compute** Modus gewechselt werden kann. Schematisch kann man sich den beschriebenen **endlichen Automaten** wie in Abblidung 2.3 vorstellen, wobei die Transitionen den jeweiligen Benutzereingaben entsprechen. Aus Gründen der Übersicht wurde der Wechsel in den **Compute** Modus weggelassen.

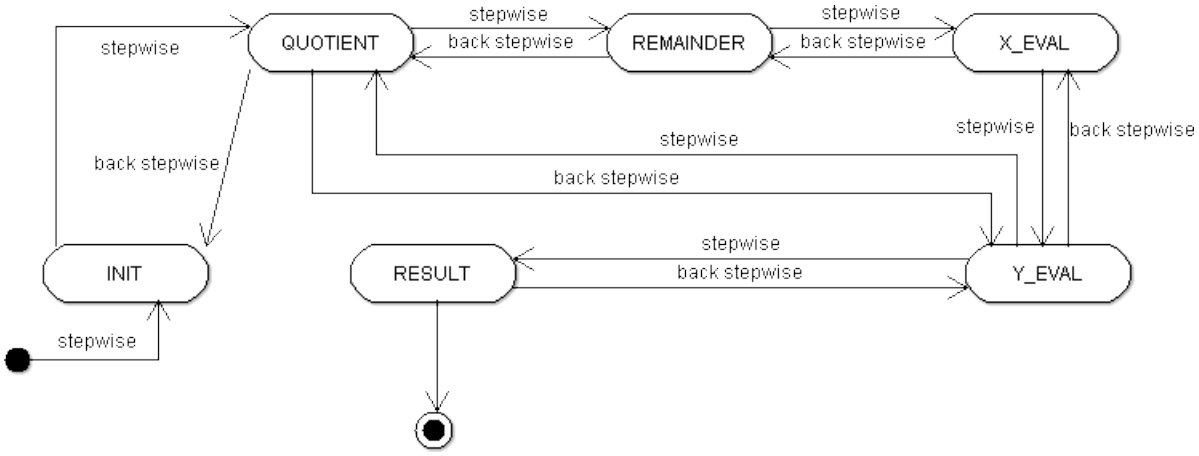

Abbildung 2.3: Der stepwise Modus als endlicher Automat

2.3.2 Der Compute Modus

Der **Compute** Modus ist im Gegensatz zum **Stepwise** Modus einfacher aufgebaut. Durch betätigen des Buttons **Compute** wird die Berechnung des **Extended Euclidean** durchgeführt, falls die Eingabefelder nicht leer sind. In diesem Fall wird, wie auch im **Stepwise** Modus, die Funktion `xeuclid.xeuclid(p, q)` aufgerufen und die Ergebnisse der Berechung mit den entsprechenden **Getter** Methoden zwischengespeichert. Danach wird die Ausgabetabelle zeilenweise mit den Zwischenergebnissen der Berechnung gefüllt. Das Endergebnis und die Berechnungsvorschrift werden unten angezeigt. Zusätzlich wird im Visualisierungsfeld das Endergebnis angezeigt und **blau** eingefärbt. In der Tabelle wird im **Compute** Modus auf farbige Hervorhebung verzichtet, bis auf das Endergebnis, das auch in der Tabelle wiederum in **blau** dargestellt wird.

2.3.3 Die Exportfunktion

Nach Durchführung einer Berechnung ist es möglich, das Ergebnis zu exportieren. Man hat die Wahl das Ergebnis in einer LaTeX-, PDF- oder CSV-Datei zu exportieren. Die Export-Funktionalität ist in der Klasse `FileExporter` im Package `org.jcryptool.visual.xeuclidean.export` gekapselt. Die Exportfunktion ist im Menü untergebracht und muss zuerst registriert werden. Dies geschieht mit der `createActions()` Methode in der Klasse `XEuclideanView`. Es wird jeweils ein Objekt der abstrakten Klasse `Action` erzeugt und die Methode `run()` überschrieben. In dieser Methode wird mit einem `FileDialog` der Speicherort der Exportdatei ausgewählt. Danach wird ein `FileExporter` erzeugt, der je nachdem von welcher `Action` es ausgelöst wurde, entscheidet, welche Ausgabedatei erzeugt wird. Der Konstruktor des `FileExporters` konsumiert als Parameter das `xeuclid` Objekt, das dem Algorithmus entspricht, und eine Referenz zur Ausgabedatei ist. Danach wird entweder die Methode `exportToPDF()`, `exportToLatex()` oder `exportToCSV()` aufgerufen. Die Methode `exportToPDF()` nutzt die `iText`[4] Bibliothek, um eine PDF-Datei zu erstellen. Es wird ein `PDFWriter` erzeugt, der die Ausgabetabelle enthält. Die Tabelle wird mit der Methode `addCell()` gefüllt, beschrieben in [Low06].
Die Erzeugung der LaTeX- und CSV-Datei erfolgt durch die `write()` Methode der Klasse `OutputStreamWriter`. Im Gegensatz zur CSV-Datei wird für den LaTeX-Export die LaTeX-Syntax verwendet. Die CSV-Datei enthält die Ergebnisse der Berechnung in Textform. Als Seperator wird das Semikolon gewählt.

[4] Eine freie Java Bibliothek, um PDF Datein zu generieren: http://www.lowagie.com/iText/

2.4 Die Funktionsweise des Plug-ins

Über das Menü **Visuals** kann das **Extended Euclidean** Plug-in (Abbildung 2.4) ausgewählt werden. Dadurch wird das Plug-in geladen und in seiner Ansicht maximiert.

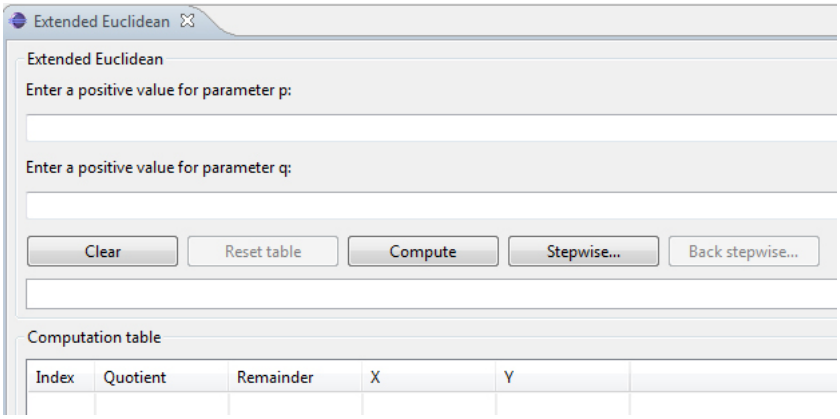

Abbildung 2.4: Das Extended Euclidean Plug-in

In die Eingabemaske **p** und **q** (Abbildung 2.5) können zwei beliebige große positive Zahlen eingegeben werden.

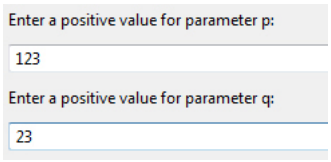

Abbildung 2.5: Eingabe der Parameter

Danach stehen verschiedene Auswahlmöglichkeiten (Abbildung 2.6) zur Verfügung:

- **Clear** - löscht die aktuelle Eingabe.
- **Reset table** - löscht nur die Tabelle, die aktuelle Eingabe bleibt erhalten (zu Beginn noch nicht auswählbar).
- **Compute** - führt die Berechnung aus.
- **Stepwise...** - die Berechnung wird schrittweise durchgeführt. Es wird der nächste Schritt in der Berechnung durchführt.
- **Back stepwise...** - die Berechnung wird schrittweise durchgeführt. Es wird der vorhergehende Schritt ausgeführt (zu Beginn noch nicht auswählbar).

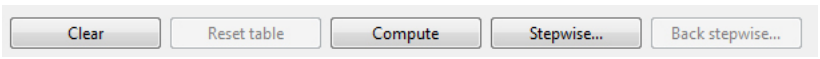

Abbildung 2.6: Verschiedene Auswahlmöglichkeiten

Das Ergebnis der Berechnung wird im **Visualisierungsfeld** dargestellt (Abbildung 2.7).

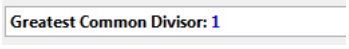

Abbildung 2.7: Das Visualisierungsfeld für das Endergebnis

Ferner werden in der **Berechnungstabelle** alle Teilschritte der Berechnung vollständig als Tabelle angezeigt, sofern man sich für die **Compute** Variante entschieden hat (Abbildung 2.8).

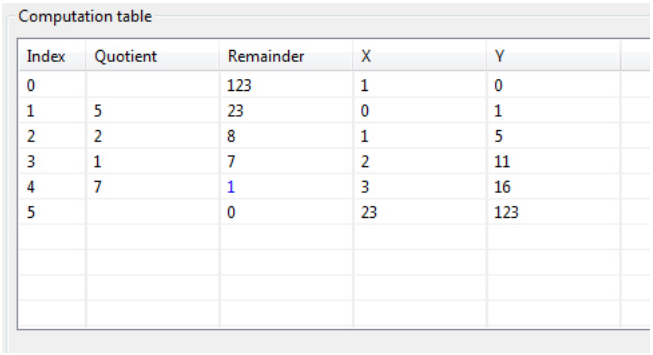

Abbildung 2.8: Berechnungstabelle des Extended Euclidean

Die Tabelle beinhaltet fünf Spalten: **Index, Quotient, Remainder, X** und **Y**. Das Endergebnis ist **blau** dargestellt. Zusätzlich wird unten die Berechnungsvorschrift: $gcd(p,q) = px + qy$ und die dazugehörige Berechnung angezeigt (Abbildung 2.9).

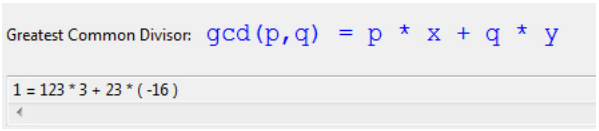

Abbildung 2.9: Die Berechnungsvorschrift und das Endergebnis

Falls man sich für die **Stepwise** Variante entschieden hat, wird in dem **Visualisierungsfeld** die aktuell durchzuführende Berechnung farbig angezeigt (Abbildung 2.10).

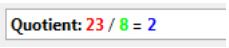

Abbildung 2.10: Das Visualisierungsfeld für den aktuellen Berechnungsschritt

Zusätzlich wird in dem **Berechnungstabelle** Bereich, abhängig von der aktuellen Position der Berechnung, die Tabelle schrittweise ausgefüllt. Dabei stimmen die Farben mit den Farben des **Visualisierungsfeldes** überein. Mit der Farbe **blau** wird immer das Ergebnis der aktuellen Berechnung dargestellt (Abbildung 2.11).

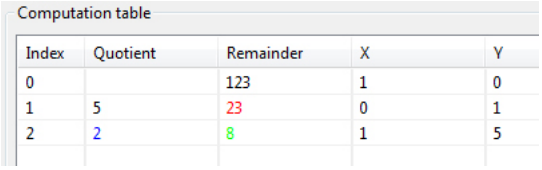

Abbildung 2.11: Teilschritte der Tabelle

Nun kann die Berechnung schrittweise mit dem Button **Stepwise...** sowohl vorwärts als auch mit dem Button **Back stepwise...** rückwärts durchlaufen werden. Falls das Ende der Berechnung erreicht ist, wird der Button **Stepwise...** deaktiviert und das Endergebnis sowohl unten als auch im **Visualisierungsfeld** angezeigt. Es ist auch möglich, während man

die Berechnung schrittweise durchläuft, auf den Butten **Compute** zu klicken, um die schrittweise Berechnung abzubrechen und die Berechnung vollständig bis zum Ende durchzuführen.

Nach Durchlauf einer Berechnung kann man das Ergebnis in die drei Formate PDF, LaTeXund CSV exportieren (Abbildung 2.12).

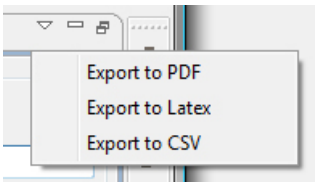

Abbildung 2.12: Export in PDF, CSV oder LaTeX

3 Das Chinese Remainder Theorem Plug-in

Das zweite Plug-in in dieser Diplomarbeit ist das **Chinese Remainder Theorem** Plug-in. Der Algorithmus wird verwendet, um simultane Kongruenzen zu lösen. Er wird z.B. benutzt, um den RSA-Algorithmus zu beschleunigen, speziell für die Erzeugung digitaler Signaturen und die Entschlüsselung. Eine weitere Anwendung des **Chinese Remainder Theorem** ist die Berechnung der Wurzel bei dem **Fiat-Shamir** Verfahren, auf das in dieser Diplomarbet nicht weiter eingegangen wird.

Wiederum gestalten wir das Plug-in einfach, damit der Algorithmus dem Benutzer schrittweise erläutert wird. In Abschnitt 3.1 beschreiben wir mathematisch den **Chinese Remainder Theorem** Algorithmus. Der Entwurf des Plug-ins wird in Abschnitt 3.2 dargestellt, die detailierte Implementierung in Abschnitt 3.3 beschrieben und der Abschnitt 3.4 widmet sich dann der Funktionsweise des Plug-ins.

3.1 Der Chinese Remainder Theorem Algorithmus

Der **Chinese Remainder Theorem** Algorithmus ist eine Lösungsmethode, um simultane Kongruenzen zu lösen. Wir haben Gleichungen der Form

$$x \equiv a_0 \ mod \ m_0, \qquad x \equiv a_1 \ mod \ m_1, \qquad \dots \qquad x \equiv a_n \ mod \ m_n \tag{3.1}$$

vorliegen, wobei $m_1, \dots, m_n$ natürliche Zahlen, die paarweise zueinander teilerfremd und $a_1, \dots a_n$ ganze Zahlen sind. Nun wird

$$m = \prod_{i=0}^{n} m_i, \qquad M_i = m/m_i, \qquad 0 \leq i \leq n \tag{3.2}$$

berechnet. Es gibt eine eindeutige Lösung der simultanen Kongruenzen. Es gilt die Gleichung

$$gcd(m_i, M_i) = 1, \qquad 0 \leq i \leq n \tag{3.3}$$

da die m_i paarweise teilerfremd sind. Wir benutzen den **Extended Euclidean**, um die Zahlen $y_i \in \mathbb{Z}, 0 \leq i \leq n$ zu berechnen mit

$$y_i M_i \equiv 1 \ mod \ m_i, \qquad 0 \leq i \leq n. \tag{3.4}$$

Nun können wir

$$x = (\sum_{i=0}^{n} a_i y_i M_i) \ mod \ m \tag{3.5}$$

setzen und erhalten eine Lösung x der simultanen Kongruenz. Wir übernehmen folgendes Theorem aus [Buc08].

Theorem 3.1.1 *Seien $m_0, \dots, m_n$ paarweise teilerfremde natürliche Zahlen und seien $a_0, \dots a_n$ ganze Zahlen. Dann hat die simultane Kongruenz eine Lösung x, die eindeutig ist mod $m = \prod_{i=0}^{n} m_i$.*

Den Beweis dazu kann man in [Buc08] nachlesen. Wir wollen den Algorithmus anhand eines Beispiels zeigen.
Beispiel Wir wollen die simultane Kongruenz

$$x \equiv 2 \ mod \ 4, \qquad x \equiv 1 \ mod \ 3, \qquad x \equiv 4 \ mod \ 5, \qquad x \equiv 3 \ mod \ 7$$

lösen. Also ist $m_0 = 4,\ m_1 = 3,\ m_2 = 5,\ m_3 = 7,\ a_0 = 2,\ a_1 = 1,\ a_2 = 4,\ a_3 = 3$. Dann ist $m = 420,\ M_0 = 420/4 = 105,\ M_1 = 420/3 = 140,\ M_2 = 420/5 = 84,\ M_3 = 420/7 = 60$. Wir lösen $y_0 M_0 \equiv 1\ mod\ m_1,\ d.h.\ y_1 \equiv 1\ mod\ 4$. Die kleinste nicht negative Lösung ist $y_0 = 1$. Wir lösen $y_1 M_1 \equiv 1\ mod\ m_1,\ d.h.\ y_1 \equiv 1\ mod\ 3$. Die absolut kleinste Lösung ist $y_1 = -1$. Wir lösen $y_2 M_2 \equiv 1\ mod\ m_2,\ d.h. y_2 \equiv 1\ mod\ 5$. Die absolut kleinste Lösung ist $y_2 = -1$. Schließlich lösen wir $y_3 M_3 \equiv 1\ mod\ m_3,\ d.h. y_3 \equiv 1\ mod\ 7$. Die kleinste nicht negative Lösung ist $y_3 = 2$. Daher erhalten wir $x \equiv 210 - 140 - 336 + 360 \equiv 94\ mod\ 420$. Eine Lösung der simultanen Kongruenz ist $x = 94$.

Man beachte, dass in dem beschriebenen Algorithmus die Zahlen y_i und M_i nicht von den Zahlen a_i abhängen. Sind die Werte y_i und M_i berechnet, dann kann man die Formel $x = (\sum_{i=0}^{n} a_i y_i M_i)\ mod\ m$ benutzen, um die simultane Kongruenz für jede Auswahl von Werten a_i zu lösen.

3.2 Der Entwurf des Plug-ins

Das Plug-in ist eine Implementierung einer **View** und gliedert sich in 5 Packages:

- `org.jcryptool.visual.crt`
- `org.jcryptool.visual.crt.algorithm`
- `org.jcryptool.visual.crt.export`
- `org.jcryptool.visual.crt.test`
- `org.jcryptool.visual.crt.views`

Im Package `org.jcryptool.visual.crt` ist die **Activator** Klasse `ChineseRemainderTheoremPlugin` enthalten, die den Lebenszyklus des Plug-ins steuert. Das Package `org.jcryptool.visual.algorithm` enthält den **Chinese Remainder Theorem** Algorithmus, der vollständig gekapselt von der **View** ist, die im Package `org.jcryptool.visual.crt.views` vorzufinden ist. Die Exportfunktion ist in das Package `org.jcryptool.visual.crt.export` ausgelagert und die JUnit Testklasse, die aber, wie auch bei dem Plug-in **Extended Euclidean**, nicht für die Ausführng des **Chinese Remainder Theorem** Plug-ins nötig ist, ist in dem Package `org.jcryptool.visual.crt.test` vorzufinden.

Das Plug-in benötigt für die Ausführung das **Extended Euclidean** Plug-in. Dies ist in der `MANIFEST.ML` unter dem Punkt `Require-Bundle` aufgelistet. Die schematische Darstellung des Plug-ins ist in Abbildung 3.1 und Abbildung 3.2 abgebildet. Aus Tabelle 3.1 kann man die Beschreibung der einzelnen Elemente des User Interfaces entnehmen.

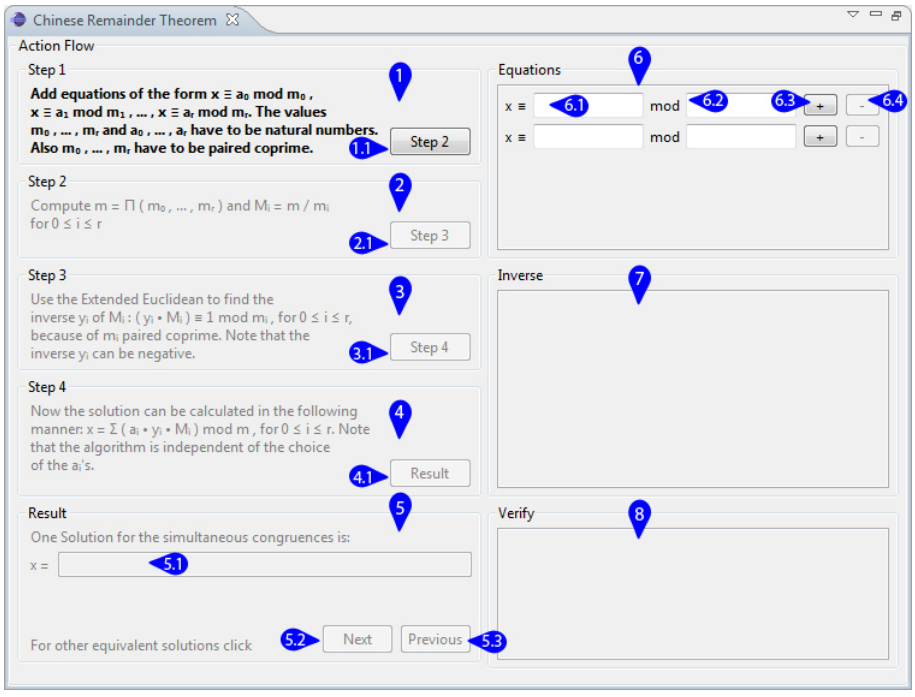

Abbildung 3.1: Das User Interface des Chinese Remainder Theorem Plug-ins

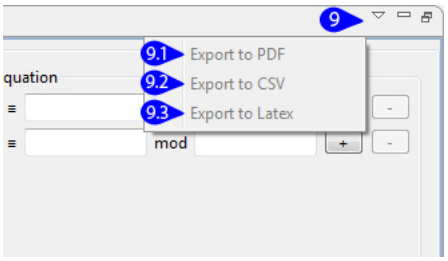

Abbildung 3.2: Das Menü des Chinese Remainder Theorem Plug-in

#	Typ	Initialer Zustand	Text	Beschreibung
1	Group	aktiviert	siehe Abbildung 3.1	Beschreibung des ersten Schrittes.
1.1	Button	aktiviert	Step 2	Wechselt in Group **Step 2**.
2	Group	deaktiviert	siehe Abbildung 3.1	Beschreibung des zweiten Schrittes.
2.1	Button	deaktiviert	Step 3	Wechselt in Group **Step 3**.
3	Group	deaktiviert	siehe Abbildung 3.1	Beschreibung des dritten Schrittes.
3.1	Button	deaktiviert	Step 4	Wechselt in Group **Step 4**.
4	Group	deaktiviert	siehe Abbildung 3.1	Beschreibung des vierten Schrittes.
4.1	Button	deaktiviert	Result	Wechselt in Group **Result**.
5	Group	deaktiviert	siehe Abbildung 3.1	Ausgabe des Endergebnisses.
5.1	Textfield	deaktiviert	leer	Ausgabe des Endergebnisses.
5.2	Button	deaktiviert	Next	Ausgabe der nächsten Lösung für die simultane Kongruenz.
5.3	Button	deaktiviert	Previous	Ausgabe der vorherigen Lösung für die simultane Kongruenz. Falls die vorherige Lösung negativ ist, wird der Button deaktiviert.
6	Group	aktiviert	siehe Abbildung 3.1	Eingabe der Gleichungen.
6.1	Textfield	aktiviert	leer	Eingabe des Parameters a_i.
6.2	Textfield	aktiviert	leer	Eingabe des Parameters m_i.
6.3	Button	aktiviert	+	Hinzufügen einer neuen Gleichung.
6.4	Button	deaktiviert	-	Entfernen der Gleichung. Die minimale Anzahl von Gleichungen ist zwei.
7	Group	deaktiviert	leer	Ausgabe der Zwischenschritte.
8	Group	deaktiviert	leer	Ausgabe der Lösung mit Gleichung, um das Ergebnis zu verifizieren.
9	Menu	deaktiviert	∇	Hier kann nach Durchführung der Berechnung das Ergebnis als **PDF**, **CSV** oder **LaTeX** Dateiformat exportiert werden. Das Menü wird erst nach Durchführen einer Berechnung aktiv.
9.1	Menu Item	deaktiviert	Export to PDF	Ergebnis der Berechnung als **PDF** Datei exportieren.
9.2	Menu Item	deaktiviert	Export to CSV	Ergebnis der Berechnung als **CSV** Datei exportieren.
9.3	Menu Item	deaktiviert	Export to LaTeX	Ergebnis der Berechnung als **LaTeX** Datei exportieren.

Tabelle 3.1: Die Verhaltensbeschreibung des Chinese Remainder Theorem Plug-ins

3.3 Die Implementierung

Die Klasse `ChineseRemainderTheoremView` repräsentiert die **View** des Plug-ins. Sie erzeugt das Menü und registriert die Methoden für die Exportfunktion. Die Exportfuntionen sind, bis auf die Erzeugung der Ausgabedatei, weitgehend aus dem **Extended Euclidean** Plug-in übernommen. Der komplette Aufbau der **View** wurde in eine seperate Klasse `CRTGroup` ausgegliedert, um die Übersichtlichkeit zu erhöhen.

Die Klasse `CRTGroup` erbt von `org.eclipse.swt.widgets.Group` und implementiert das Interface `Constants`. Das Interface `Constants` dient lediglich dazu, konstante Strings, Unicode Zeichen und Schriften, die wir für Beschreibungen brauchen, zentral in einer Klasse zu bündeln. Somit können Änderungen der Namen des User Interfaces oder Testpassagen im Plug-in einfach vorgenommen werden, ohne die Position in der Klasse `CRTGroup` abzusuchen.

Die `CRTGroup` hat einen zwei Spalten Aufbau. In der linken Spalte ist der Ablauf des Plug-ins dargestellt und in der rechten Spalte werden die Gleichungen aufgestellt und die Zwischenergebnisse angezeigt. Jeder Ablaufschritt in der linken Spalte besteht aus einem Titel, einer Textpassage, der Beschreibung des aktuellen Schrittes, und einem Button, um zum nächsten Schritt zu wechseln. Nur der letzte Schritt, der für das Darstellen der Lösung ist, hat zusätzlich ein Textfeld, in dem das Ergebnis nach Beendigung des Algorithmus steht und zwei Buttons, um sich weitere Lösungen anzeigen zu lassen. Die rechte Spalte enthält drei Gruppen: **Equation**, **Inverse** und **Verify**.

Das UML-Aktivitätsdiagramm zum **Chinese Remainder Theorem** Plug-in ist in Abbildung 3.3 dargestellt. Der detailierte Aufbau der `equationGroup` ist in Abschnitt 3.3.1 erklärt. Nach Aufstellung der Gleichungen wechselt man mit dem Button **Step 2** zum nächsten Schritt, wenn die Eingabefelder in der `equationGroup` nicht leer sind. Falls die Eingabe fehlerhaft ist, zum Beispiel wenn man zweimal dasselbe Modul verwendet oder wenn die Moduli nicht paarweise teilerfremd sind, öffnet sich das Dialogfeld `CheckingEquationDialog`. Die Klasse `CheckingEquationDialog` erbt von der **JFace** Klasse `TitleAreaDialog` und dient dazu, eine Fehleingabe der Gleichungen zu korrigieren. Dabei werden alle Gleichungen und Werte in das Dialogfeld übernommen. Es sind nur die Modulfelder aktiv, die nicht paarweise teilerfremd sind oder in dem das Modul gleich ist. Alle anderen Eingabefelder sind inaktiv und können nicht verändert werden. Desweiteren gibt es einen Button **Suggestion** und einen Button **Apply**.

Die Idee des **Suggestion** Buttons ist es, einen gültigen Wert vorzuschlagen, den der Benutzer dann entweder mit dem **Apply** Button einloggen oder sich weiter Werte vorschlagen lassen kann. Es wird ein `Mouselistener` verwendet, der als Parameter einen `MouseAdapter` übergeben bekommt. In diesem `MouseAdapter` wird die Methode `mouseUp` überschrieben. Zuerst wird die Position der Gleichungszeile, von der der Button gedrückt wurde, ermittelt. Danach wird der `VerifyListener` vom Modulo Textfeld entfernt, da man ansonsten nicht in das Textfeld schreiben kann. Der vorhandene Wert wird um Eins erhöht, in der Variable `suggestionValue` gespeichert und ausgehend davon der nächste gültige Wert ermittelt. Dazu benutzen wir eine verschachtelte Schleife, die aus einer äußeren `while` und einer inneren `for` Schleife besteht, um alle Werte der Modulo Textfelder zu durchlaufen. Es wird nun mit dem **Extended Euclidean** ein Wert gesucht, dessen Wert nicht gleich den anderen Werten der Textfelder sind oder dessen größter gemeinsamer Teiler ungleich Eins ist. In diesem Fall wird die Variable `suggestionValue` um Eins erhöht und die äußere Schleife wieder von vorne durchlaufen, bis ein Wert gefunden wurde, dessen gcd gleich Eins ist. Der auf diese Weise gefundene Wert wird in das Textfeld geschrieben und der `VerifyListener` wieder angefügt. Die dazugehörige verschachtelte Schleife des Quellcode ist in Listing 3.1 dargestellt.

```
1  suggestionValue = new BigInteger(textfieldMSet[i].getText());
2  suggestionValue = suggestionValue.add(BigInteger.ONE);
3  XEuclid gcd = new XEuclid();
4  boolean search = true;
5  while (search) {
6     for (int k = 0; k < textfieldMSet.length; k++) {
7        BigInteger tmpGcd = gcd
8              .xeuclid(new BigInteger(textfieldMSet[k].getText()), suggestionValue);
9
10       if (new BigInteger(textfieldMSet[k].getText()).compareTo(suggestionValue) == 0
11             || tmpGcd.compareTo(BigInteger.ONE) != 0) {
12
13          suggestionValue = suggestionValue.add(BigInteger.ONE);
14          search = true;
15          break;
16       } else {
```

```
            search = false;
        }
    }
}
```

Listing 3.1: Schleife des Suggestion Buttons

Der **Apply** Button dient dazu, die Benutzereingabe oder den vom Plug-in vorgeschlagenen Wert zu übernehmen und einzuloggen. Dazu verwenden wir wieder einen `MouseListener`, der als Parameter ein `MouseAdapter` übergeben bekommt. In diesem `MouseAdapter` wird die Methode `mouseUp` überschrieben. Zuerst wird die Position der Gleichung ermittelt, von der der **Apply** Button gedrückt wurde. Wir erzeugen uns zusätzlich ein Array `marking` vom Typ `int`, in dem wir uns merken, welche Werte gültig sind oder nicht, wobei die 0 für einen gültigen Wert und die -1 für einen ungültigen Wert steht. Die Werte in dem Array `marking` werden mit 0 initialisiert. Nun gehen wir in einer Schleife die Menge der Modul Textfelder durch und vergleichen jeden Wert mit den anderen Werten der Textfelder, ob der größte gemeinsame Teiler ungleich Eins ist. Dazu benutzen wir die `XEuclid` Klasse des **Extended Euclidean** Plug-ins. Falls der größte gemeinsame Teiler ungleich Eins ist, wird die Position im Array `marking` mit -1 markiert. Desweiteren wird noch auf Gleichheit der beiden Textfelder verglichen. Auch in diesem Fall wird die Position im Array mit -1 markiert. Der Quellcode der Schleife ist in Listing 3.2 dargestellt.

```
for (int i = 0; i < textfieldMSet.length; i++) {
    BigInteger a = new BigInteger(textfieldMSet[i].getText());
    for (int j = i + 1; j < textfieldMSet.length; j++) {
        XEuclid gcd = new XEuclid();
        BigInteger tmpValue = gcd.xeuclid(a, new BigInteger(textfieldMSet[j].getText()));
        if (tmpValue.compareTo(BigInteger.ONE) != 0) {
            marking[j] = -1;
            if (a.compareTo(new BigInteger(textfieldMSet[j].getText())) == 0) {
                marking[i] = -1;
            }
        }
    }
}
```

Listing 3.2: Schleife des Verify Buttons

Falls das Textfeld nun positiv verifiziert wurde, können wir das Textfeld, den **Apply** Button und den **Suggestion** Button deaktivieren. Wenn alle **Apply** Buttons deaktiviert sind, wird der **Ok** Button des Dialogfeldes aktiviert und man kann den Dialog mit Bestätigen des **Ok** Buttons verlassen.

Nach erfolgreichem Schließen des Dialogfeldes wird ein `ChineseRemainderTheorem` Objekt erzeugt, das den Algorithmus entspricht. Mit dem Aufruf `BigInteger result = crt.crt(moduli, a)` werden dem Algorithmus die Parameter für die Berechung des **Chinese Remainder Theorems** übergeben und das Endergebnis in der Variable `result` abgespeichert, dabei sind die Werte für die a_is und m_is in den Arrays `a` und `moduli` vorzufinden.

Von **Step 2** gelangt man durch Drücken des Buttons **Step 3** zum dritten Schritt. In der **Inverse** Gruppe werden die ersten Zwischenergebnisse angezeigt. Die Klasse `ChineseRemainderTheorem` bietet dabei die Methode `crt.getBigM()`, die ein `BigInteger` Array zurückliefert, in dem die Werte der M_is vorzufinden sind. Dieses Array wird nun in einer Schleife durchlaufen und die Werte für die M_is und die Werte für die Inversen Elemente y_i werden in die Gruppe `inverseGroup` geschrieben. Dabei werden die Werte für die inversen Elemente erst noch ausgeblendet, da diese im vierten Schritt dargestellt werden. Durch Klick auf den Button **Step 4** gelangt man zum vierten Schritt und es werden die zuvor ausgeblendeten inversen Elemente der Berechnung angezeigt. Die Beschreibung des nächsten Schrittes wird nun in `Step 4` angezeigt und hervorgehoben. Zum Endergebnis gelangt man nun durch Drücken des Button **Result**. Dies bewirkt zum einen, dass das Endergebnis in der `resultGroup` angezeigt wird und zum anderen, dass die `verifyGroup` aufgebaut wird. Auch wird der Button `next` aktiviert, um weitere Lösungen der simulanten Kongruenzen anzuzeigen. Die `verifyGroup` dient dem Benutzer als Hilfe dafür, um sich die kompletten Gleichungen mit Lösung anzuzeigen. Mit dem **Next** Button in der `resultGroup` kann die nächste Lösung angezeigt werden. Dabei werden auch parallel die Werte der `verifyGroup` angepasst. Mit dem **Previous** Button kann man sich die nächst kleinere Lösung anzeigen lassen, dabei ist zu beachten, dass nur positive Lösungen angezeigt werden. Falls die nächste Lösung negativ wird, wird der **Previous** Button deaktiviert.

Der Ablauf der Berechnung ist von jedem Schritt aus durch Drücken der **Plus** oder **Miunus** Buttons in der **Equation** Gruppe abbrechbar. Dies bewirkt ein Reset der Werte und eine Reinitialisierung der View. Dabei wird entweder eine neue Gleichungszeile eingefügt oder entfernt. Die Klasse `CRTGroup` bietet dazu die Methode `reset()` an, in der das Reset vorgenommen wird. Als weitere Hilfsmethode wird die Methode `convertToSubset(int id)` angeboten, die den Wert von `id` als **Subscript** Unicode[1] Zeichen umwandelt. Dabei wird `id` Zeichen für Zeichen in einer Schleife durchlaufen und der entsprechende `Character` Wert in ein Unicode Zeichen konvertiert.

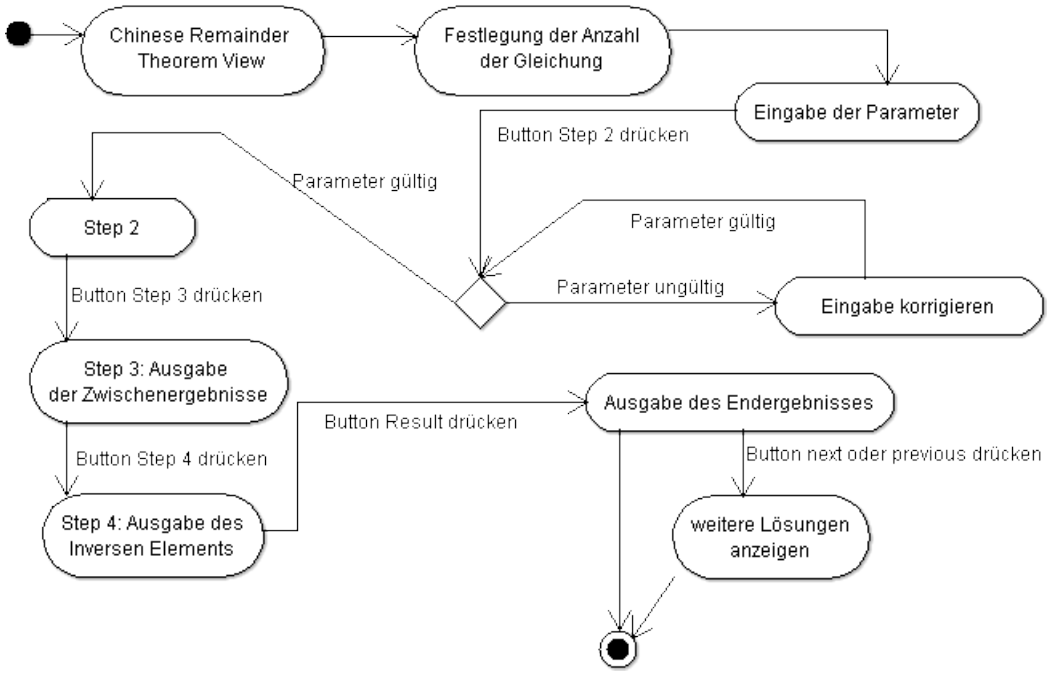

Abbildung 3.3: UML Aktivitätsdiagramm des Chinese Remainder Theorem Plug-in

3.3.1 Aufbau der Gleichungen für die simultanen Kongruenzen

Um den Erzeugungsprozess der Gleichungen zu vereinfachen, haben wir die Klassen `Equations` und `Equation` eingeführt. Die Struktur ist dem Entwurfsmuster **Composite** aus [EG09] nachempfunden. Dabei ist die Klasse `Equations` als Menge für die Gleichungen anzusehen. Sie enthält einen Vector `equationSet` vom Typ `Equation`, in dem wir unsere Gleichungen aufbewahren. Desweiteren wird mit der Methode `createEquation` ein `Equation` Objekt, mit der dazugehörigen `ID` erzeugt und mit `e.addEquationToGroup()` der Gruppe `equationGroup` angefügt, der dazugehörige Quellcode ist in Listing 3.3 dargestellt. Weitere Methoden sind, `removeEquation(Equation e)` zum Entfernen einer Gleichung, `getIndexOf(Equation e)`, um die Position der Gleichung `e` zu erhalten, `getNumberOfEquations()`, um die Anzahl der Gleichungen im Vector `equationSet` zu ermitteln, `lastElement()`, um die letzte Gleichung zu erhalten und `getEquationSet()`, um den Vector `equationSet` für andere Klassen zur Verfügung zu stellen.

```
1 public void createEquation(int id, Group equationGroup, CRTGroup mainGroup) {
2     Equation e = new Equation(id, this, equationGroup, mainGroup);
3     equationSet.add(e);
4     e.addEquationToGroup();
5 }
```

Listing 3.3: Erzeugung einer Gleichung

Die Klasse `Equation` repräsentiert eine Gleichung. Dabei erzeugt die Klasse eine ganze Gleichungszeile mit entsprechenden Labels, Textfields und Buttons. Eine Gleichungszeile besteht aus dem Label **x**, einem Label ≡, einem Textfield `private Text textfieldA`, dem Label **mod**, dem Textfield `private Text textfieldM` und den Buttons **Plus** und **Minus**. Für die beiden Textfields wurden, wie auch beim **Extended Euclidean**, ein `VerifyListener` verwendet, die nur

[1] Ist ein internationaler Standard, in dem für jedes Schriftzeichen oder Textelement ein digitaler Code festgelegt wird: http://www.unicode.org

Zahlen von 0 bis 9 zulassen und führende Nullen verweigern. Die genaue Beschreibung kann im Abschnitt 2.3 nachgelesen werden. Der **Plus** Button bewirkt, dass eine neue Gleichungszeile eingefügt wird. Dabei wird die neue Gleichung unmittelbar nach der Zeile, von der aus der Button gedrückt wurde, eingefügt. Dies bedeutet, dass wenn wir fünf Gleichungen haben und wir auf den **Plus** Button der zweiten Gleichung drücken, wird eine neue Gleichungszeile genau zwischen die zweite und dritte Gleichung eingefügt. Dabei werden eventuelle Eingabewerte, die schon vorhanden sind, mitgeführt und nicht gelöscht. Der Quellcode dazu ist in Listing 3.4 abgebildet. Um dies zu erreichen, fügen wir dem `plusButton` einen `MouseListener` hinzu. Der `MouseListener` bekommt als Parameter einen `MouseAdapter`, dessen Methode `mouseUp(MouseEvent e)` überschrieben wird. In der Variable `instance` merken wir uns das aktuelle `Equation` Objekt, von dem aus der **Plus** Button gedrückt wurde. Ferner wird die Position der aktuellen Gleichungszeile ermittelt und in der Variable `tmpIndex` gespeichert. Es wird mit `createEquation` eine neue Gleichungszeile erzeugt und an die Position `tmpIndex + 1` angefügt. Im nächsten Schritt werden alle Werte der Eingabefelder ab der Position `tmpIndex + 1` um einen Schritt nach unten verschoben. Dabei müssen wir zuerst die `VerifyListener` der Textfields entfernen, da man sonst nicht in die Textfields schreiben kann. Um die Werte zu kopieren, gehen wir in einer Schleife von unten nach oben vor, damit man keine Werte überschreibt. Danach werden die zuvor entfernten `VerifyListener` wieder hinzugefügt.

```
1  private Button plusButton;
2  plusButton = new Button(equationGroup, SWT.NONE);
3  plusButton.setLayoutData(new GridData(30, 20));
4  plusButton.setText("+");
5  plusButton.addMouseListener(new MouseAdapter() {
6     @Override
7     public void mouseUp(MouseEvent e) {
8        if (CRTGroup.execute){
9           mainGroup.reset();
10       }
11       Vector<Equation> equationSet = equations.getEquationSet();
12       int tmpIndex = equationSet.indexOf(instance);
13       equations.createEquation(tmpIndex + 1, equationGroup, mainGroup);
14
15       for (Equation equation : equationSet) {
16          equation.textfieldA.removeVerifyListener(equation.aTextfieldVerifyListiner);
17          equation.textfieldM.removeVerifyListener(equation.mTextfieldVerifyListiner);
18       }
19       for (int i = equationSet.size() - 1; i > tmpIndex; i--) {
20          equationSet.get(i).setTextfieldA(equationSet.get(i - 1).getTextfieldA());
21          equationSet.get(i).setTextfieldM(equationSet.get(i - 1).getTextfieldM());
22       }
23       equationSet.get(tmpIndex + 1).setTextfieldA("");
24       equationSet.get(tmpIndex + 1).setTextfieldM("");
25
26       for (Equation equation : equationSet) {
27          equation.textfieldA.addVerifyListener(equation.aTextfieldVerifyListiner);
28          equation.textfieldM.addVerifyListener(equation.mTextfieldVerifyListiner);
29       }
30    }
31 });
```

Listing 3.4: Implementierung des Plus Buttons

Der `Minus` Button bewirkt im Gegensatz zum `Plus` Button, dass die Gleichungszeile, von der aus der Button gedrückt wurde, entfernt wird. Die Implementierung des `Minus` Buttons ist in Listing 3.5 abgebildet. Zuerst werden die einzelnen Elemente der Gleichungszeile auf `dispose` gesetzt, falls mehr als zwei Gleichungen vorhanden sind. Dies signalisiert dem Compiler, dass diese Elemente nicht mehr gültig sind und bei dem nächsten Neuzeichnen der Gruppe gelöscht werden können. Wir überschreiben wieder die `mouseUp` Methode des `MouseAdapters`. Bei dem **Minus** Button reicht es, die aktuelle Gleichungszeile mit der Methode `equations.removeEquation(instance)` aus dem Vector `equationSet` zu entfernen und die `equationGroup` neu zuzeichnen, was mit der Methode `equationGroup.pack()` angestoßen wird. Falls wir nur zwei Gleichungszeilen haben, müssen die `Minus` Buttons der Gleichungen deaktiviert werden, da das Minimum an Gleichungen zwei ist.

```
private Button minusButton;
minusButton = new Button(equationGroup, SWT.NONE);
minusButton.setLayoutData(new GridData(30, 20));
minusButton.setText("-");
minusButton.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseUp(MouseEvent e) {
        Vector<Equation> equationSet = equations.getEquationSet();
        if (CRTGroup.execute){
            mainGroup.reset();
        }
        if (equations.getNumberOfEquations() > 2) {
            dispose();

            equations.removeEquation(instance);

            if (equationSet.size() <= 2) {
                equationSet.firstElement().setEnableMinusButton(false);
                equationSet.get(1).setEnableMinusButton(false);
            }
            equationGroup.pack();
        }
    }
});
```

Listing 3.5: Implementierung des Minus Buttons

3.4 Die Funktionsweise des Plug-ins

Das **Chinese Remainder Theorem** Plug-in läßt sich über das Menü **Visuals** starten und in seiner Ansicht maximieren (Abbildung 3.4).

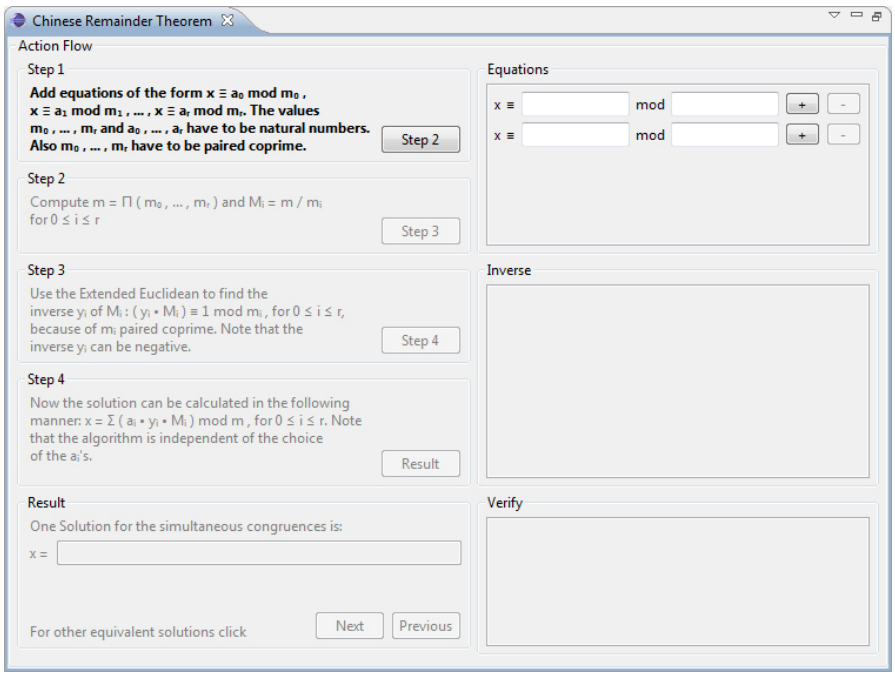

Abbildung 3.4: Das Chinese Remainder Theorem Plug-in

Das Plug-in gliedert sich in ein zwei Spalten Layout auf. Auf der linken Seite wird die Beschreibung und die Berechnungsvorschrift des aktuellen Schrittes dargestellt und auf der rechten Seite die Eingabefelder und Ergebnisse der Berechnung. In der **Equation** Gruppe auf der rechten Seite des Plug-ins können die Parameter der Gleichung eingegeben werden. Dabei können positive Zahlen beliebiger Größe verwendet werden (Abbildung 3.5).

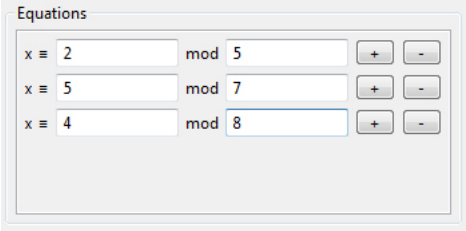

Abbildung 3.5: Eingabe der Parameter

Durch Drücken des + Buttons kann die Menge der Gleichungen erhöht und mit dem - Button verringert werden. Die minimale Anzahl der Gleichungen ist zwei. Nach Festlegung der Anzahl der Gleichungen und Eingabe der Parameter, kann man durch betätigen des Buttons **Step 2** (Abbildung 3.6) zum nächsten Schritt übergehen.

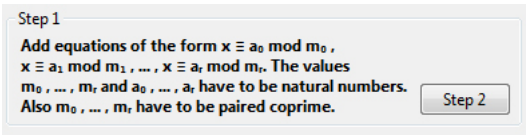

Abbildung 3.6: Der erste Schritt

Falls die Eingabeparameter der Gleichungen nicht korrekt war, erscheint ein Dialogfeld, in dem der Benutzer die Eingabe korrigieren kann (Abbildung 3.7). Der Benutzer hat die Möglichkeit, die Eingabe selbst zu korrigieren oder durch den Button **Suggestion** einen gültigen Wert vom Programm vorschlagen zu lassen. In beiden Fällen muss der Benutzer die Eingabe mit dem Button **Apply** einloggen. Mit dem **Ok** Button kehrt der Benutzer zum Programm zurück und das Dialogfeld schließt sich. Die Gruppe **Step 2** wird aktiviert und die Berechnungsvorschrift wird hervorgehoben (Abbildung 3.8).

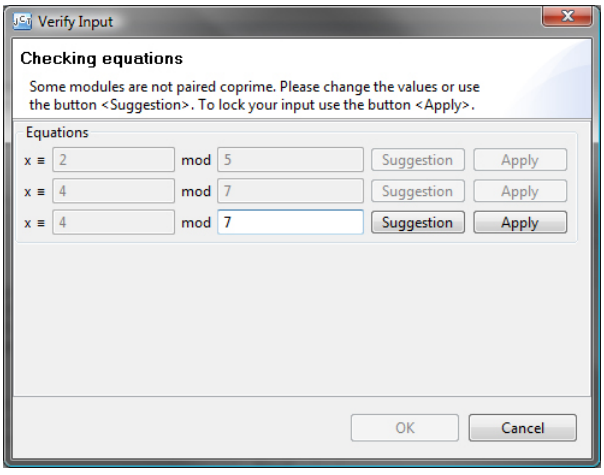

Abbildung 3.7: Dialog bei ungültiger Eingabe

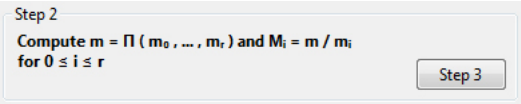

Abbildung 3.8: Der zweite Schritt

Der Benutzer kann nun durch Drücken auf Button **Step 3** zum dritten Schritt wechseln. Es werden in der **Inverse** Gruppe die Zwischenergebnisse der Berechnung angezeigt (Abbildung 3.9). Die dazu gehörige Erklärung ist zuvor in der **Step 2** Gruppe beschrieben worden. **Step 3** beschreibt die Berechnung des inversen Elements (Abbildung 3.10), die dann im nächsten Schritt **Step 4** in der **Inverse** Gruppe dargestellt wird (Abbildung 3.11).

Abbildung 3.9: Zwischenergebnisse in der Inverse Gruppe

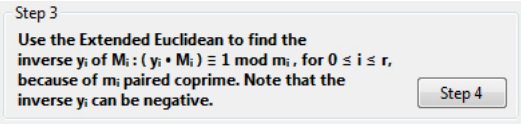

Abbildung 3.10: Der dritte Schritt

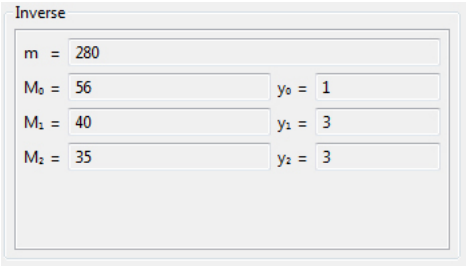

Abbildung 3.11: Die Berechnung des Inversen

Die Berechnung des Endergebnisses ist im letzten Schritt **Step 4** beschrieben (Abbildung 3.12). Das Endergebnis kann durch Drücken des **Result** Buttons nun in der Gruppe **Result** angezeigt werden (Abbildung 3.13).

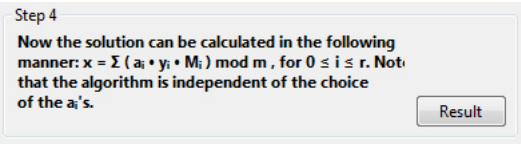

Abbildung 3.12: Die Berechnung des Endergebnisses

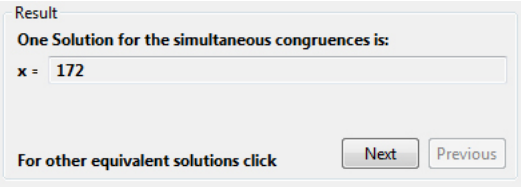

Abbildung 3.13: Darstellung des Endergebnisses

Damit der Benutzer sich von der Richtigkeit der Lösung überzeugen kann, werden in der **Verify** Gruppe alle Gleichungen noch einmal dargestellt (Abbildung 3.14).

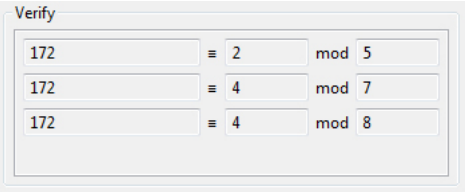

Abbildung 3.14: Verify Gruppe mit allen Gleichungen

Der Benutzer kann sich nun durch Klicken auf die Buttons **Next** und **Previous** weitere Lösungen der simultanen Kongruenzen anzeigen lassen. Dabei werden die Lösungen sowohl in der **Result** als auch in der **Verify** Gruppe dargestellt. Um eine neue Berechnung zu initiieren, reicht es, wenn man in der **Equation** Gruppe auf den **+** oder **-** Button klickt. Es wird dadurch ein Reset der Eingabefelder bis auf die **Equation** Gruppe durchgeführt und man kann weitere Gleichungen hinzufügen oder entfernen. Dieser Reset ist von jedem Schritt aus möglich.

Nach Durchlauf des Algorithmus, ist es möglich die Berechnung zu exportieren. Das Plug-in unterstützt dabei die Ausgabe als LaTeX-, PDF- oder CSV-Datei (Abbildung 3.15).

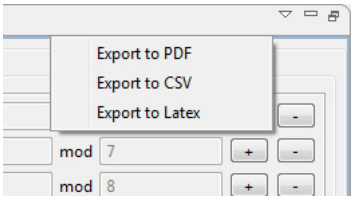

Abbildung 3.15: Export der Ausgabe

4 Das Shamir's Secret Sharing Plug-in

Das letzte Plug-in in dieser Diplomarbeit ist ein Plug-in über **Shamir's Secret Sharing**. Es ist ein von **Adi Shamir** 1979 entwickeltes Secret Sharing Verfahren, beschrieben in [Sha79], mit dem es möglich ist, ein Geheimnis auf mehrere Personen aufzuteilen, wobei eine gewisse Untermenge dieser Personen erforderlich ist, um das Geheimnis zu rekonstruieren.

Das Kapitel gliedert sich in vier Abschnitte. In Abschnitt 4.1 wird der Algorithmus mathematisch beschrieben. Der Entwurf ist aus Abschnitt 4.2 zu entnehmen und die detailierte Implementierung wird in Abschnitt 4.3 beschrieben. Zum Schluss wird in Abschnitt 4.4 die Funktionsweise des Plug-ins erklärt.

4.1 Der Shamir's Secret Sharing Algorithmus

Seien n und t natürliche Zahlen. In einem (n,t) Secret Sharing Protokoll wird ein Geheimnis auf n Personen aufgeteilt. Jede Person hat einen Teil (Share) des Geheimnisses. Wenn sich t dieser Personen zusammentun, können sie das Geheimnis rekonstruieren. Wenn sich aber weniger als t dieser Teilgeheimnisträger zusammentun, können sie keine relevante Informationen über das Geheimnis erhalten.

Der Algorithmus von Shamir basiert auf der Lagrange-Polynominterpolation und ist ein (n,t) Secret Sharing Protokoll. Alle Berechnungen werden im Ganzzahlkörper $\mathbb{Z}/p\mathbb{Z}$ durchgeführt.

Der Dealer will ein Geheimnis $s \in \mathbb{Z}/p\mathbb{Z}$ verteilen.

1. Er wählt geheime Elemente $a_j \in \mathbb{Z}/p\mathbb{Z}, 1 \leq j \leq t-1$ und konstruiert daraus das Polynom

$$a(X) = s + \sum_{j=1}^{t-1} a_j X^j. \tag{4.1}$$

   Das Polynom ist vom Grad $\leq t-1$.

2. Der Dealer berechnet die Shares $y_i = a(x_i), 1 \leq i \leq n$.

3. Der Dealer gibt dem i-ten Shareholder den Share $y_i, 1 \leq i \leq n$. Das Geheimnis ist der konstante Term $a(0)$ des Polynoms $a(X)$.

Um das Geheimnis rekonstruieren zu können, nehmen wir an, dass t Shareholder zusammen arbeiten. Ihre Shares seien $y_i = a(x_i), 1 \leq i \leq t$. Dabei ist $a(X)$ das Polynom aus 4.1. Es gilt

$$w_i = y_i \prod_{j=1, j\neq i}^{t} \frac{x_j - X}{x_j - x_i}. \tag{4.2}$$

und

$$a(X) = \sum_{i=1}^{t} w_i. \tag{4.3}$$

Dargestellt in einer Formel

$$a(X) = \sum_{i=1}^{t} y_i \prod_{j=1, j\neq i}^{t} \frac{x_j - X}{x_j - x_i}. \tag{4.4}$$

Da der Koeffizient $a(0)$ das Geheimnis ist, berechnen wir

$$s = a(0) = \sum_{i=1}^{t} y_i \prod_{j=1, j\neq i}^{t} \frac{x_j}{x_j - x_i}. \tag{4.5}$$

Diese Formel 4.5 wird von den Shareholdern benutzt, um das Geheimnis zu konstruieren.

Wir zeigen dies an einem Beispiel.

Beispiel Sei $n = 5, t = 3$. Der Dealer wählt $p = 17, x_i = i, 1 \leq i \leq 5$. Das Geheimnis sei $s = 5$. Der Dealer wählt die geheimen Koeffizienten $a_1 = 7, a_2 = 11$. Damit erhalten wir das Polynom

$$a(X) = 5 + 7X + 11X^2. \tag{4.6}$$

Die Shares sind damit $y_1 = a(1) = 6, y_2 = a(2) = 12, y_3 = a(3) = 6, y_4 = a(4) = 5, y_5 = a(5) = 9$. Diese werden nun an die Shareholder verteilt. Die Shareholder $1, 2$ und 3 werden nun benutzt, um das Geheimnis zu rekonstruieren. Es wird das Lagrange-Polynom berechnet und mit den Shares y_i gewichtet

$$w_i = y_i \prod_{j=1, j\neq i}^{3} \frac{x_j}{x_j - x_i} \qquad 1 \leq i, j \leq 3. \tag{4.7}$$

Es ergibt sich für $w_1 = 4 + 9x + 6x^2, w_2 = 3 + 2x + 11x^2, w_3 = 15 + 13x + 11x^2$. Nun bildet man die Summe der w_is und erhält das Ausgangspolynom

$$\sum_{i=1}^{3} w_i = 5 + 7x + 11x^2. \tag{4.8}$$

Der Koeffizient $a_0 = 5$ ist unser gesuchtes Geheimnis s.

4.2 Der Entwurf des Plug-ins

Das Plug-in besteht aus vier Packages:

- `org.jcryptool.visual.secretsharing`
- `org.jcryptool.visual.secretsharing.algorithm`
- `org.jcryptool.visual.secretsharing.test`
- `org.jcryptool.visual.secretsharing.views`

Wobei das Package `org.jcryptool.visual.secretsharing.test`, wie auch bei den anderen Plug-ins, aus JUnit Tests besteht, die für die Ausführung des Plug-ins nicht erforderlich sind. Das Plug-in besitzt zwei Modi, die sich in ihrer Darstellung unterscheiden. Der **Graphical** Modus dient dazu, um den Algorithmus graphisch zu visualisieren und der **Numerical** Modus, um mehr Übersicht bei großen Werten zu ermöglichen.

Das Package `org.jcryptool.visual.secretsharing.algorithm` enthält die Klasse `ShamirsSecretSharing`, die den Algorithmus implementiert. Die Klasse `Point` steht als Hilfsklasse für die Erzeugung von Punkten zur Verfügung. Sie beinhaltet die Attribute `x` und `y` vom Typ `BigInteger` und dient dazu, die Shares als x,y-Paar zusammenzufassen, um sie in dem **Graphical** Modus zu zeichnen.

4.3 Die Implementierung

In der Klasse `ShamirsSecretSharing` werden verschiedene Methoden zur Berechnung der Shares und zur Rekonstruktion zur Verfügung gestellt. Dem Konstruktor `ShamirsSecretSharing(BigInteger numberOfNecessaryPersons, BigInteger numberOfPersons, BigInteger modul, Mode mode)` werden vier Parameter übergeben. Die Anzahl der Personen, die für die Rekonstruktion nötig sind, die Anzahl der Personen, an die die Shares verteilt werden, das Modul und der Modus, indem die Berechnung ablaufen soll.

Um die Shares zu berechnen, haben wir zwei Methoden, die sich nur in ihrer Methodensignatur unterscheiden. Zum Einen wird für einen `x`-Wert eines Punktes `p` der dazu gehörige `y`-Wert berechnet und zurückgegeben und zum Anderen wird zu einer Menge von Punkten die `y`-Werte berechnet. Dabei wird unterschieden, in welchem Modus man sich befindet. Falls man sich im **Graphical** Modus befindet, wird der `y`-Wert nicht modulo dem Modul reduziert, da dies sonst den Wert für die graphische Visualisierung verfälscht. Der Quellcode für die Berechnung der Shares ist in Listing 4.1 dargestellt.

```
1  public Point[] computeShares(Point p[]) {
2     for (int i = 0; i < p.length; i++) {
3        BigInteger value = BigInteger.ZERO;
4        for (int j = coefficient.length - 1; j >= 0; j--) {
5           value = value.multiply(p[i].getX());
6           value = value.add(coefficient[j]);
7        }
8        switch (mode) {
9        case NUMERICAL:
10          p[i].setY(value.mod(modul));
11          break;
12       case GRAPHICAL:
13          p[i].setY(value);
14          break;
15       }
16    }
17    shares = p;
18    return shares;
19 }
```

Listing 4.1: Methode um die Shares zu berechnen

In einer Schleife werden die Koeffizienten des Polynoms durchlaufen und mit den `x`-Werten der Punkte multipliziert und addiert. Dabei muss vorher mit der Methode `setCoefficient(BigInteger[] coefficient)` die Koeffizienten des Polynoms gesetzt werden. Nach Berechnung der Shares, kann nun mit der Methode `interpolatePoints(Point[] pointSet, BigInteger modul)` das Lagrange Polynom (Gleichung 4.4) rekonstruiert werden. Desweiteren werden die Teilpolynome w_i (Gleichung 4.2) für die Visualisierung gespeichert.

Der struktuelle Aufbau des Plug-ins ist in dem Package `org.jcryptool.visual.secretsharing.views` zusammengefasst. Die Klasse `SecretSharingView` steuert dabei die zwei Ansichten für die verschiedenen Modi. Die Klasse besitzt jeweils eine Instanz der Klasse `ShamirsCompositeGraphical` und eine der Klasse `ShamirsCompositeNumerical`. Diese beiden Klassen sind Erweiterungen von der Klasse `Composite` und werden weiter unten beschrieben. Das Layout wird mit dem **Graphical** Modus initialisiert und kann durch den Benutzer gewechselt werden. Dabei haben beide `Composite` Klassen in der **Settings** Gruppe eine Gruppe **Select Modus**, die den aktuellen Modus anzeigt und der zwei Radio-Buttons enthält, um die Ansicht zu wechseln. Je nachdem welchen Radio-Button der Benutzer drückt, wird die Anweisung `view.showGraphical()` oder `view.showNumerical()` ausgeführt. Dadurch wird zum Beispiel das Layout mit der Anweisung `layout.topControl = shamirsCompositeGraphical` in das **Graphical** Layout gewechselt und die Ansicht für den Modus initialisiert. Die Initialisierung wird durch die Methode `adjustButtos()` durchgeführt, die alle Buttons, Textfields und sonstige Elemente auf ihren initialien Wert zurücksetzt. Der Quellcode der Methode `showGraphical()` von der Klasse `SecretSharingView` ist in Listing 4.2 abgebildet.

```
public void showGraphical() {
    if (!layout.topControl.equals(shamirsCompositeGraphical)) {
        shamirsCompositeGraphical.adjustButtons();
        layout.topControl = shamirsCompositeGraphical;
        parent.layout();
    }
}
```

Listing 4.2: Layout-Wechsel

4.3.1 Die Gruppe Select Parameter

In der Gruppe **Select Parameter** werden die nötigen Einstellungen für den Algorithmus vorgenommen. Dabei sind die Einstellungen sowohl beim **Graphical**- als auch beim **Numerical**-Modus gleich. Für die Parameter **Number of concerned persons** und **Number of persons for reconstruction** wurden Instanzen der SWT Klasse `Spinner` verwendet. Eine Instanz der `Spinner` Klasse besteht aus einem Textfeld mit zwei kleinen Pfeilen, die eine Veränderung der Wert erlaubt. Der Vorteil der `Spinner` ist es, dass man eine gewisse Abhängigkeit implementieren kann. Da die Anzahl der Personen für die Rekonstruktion nicht größer sein darf als die Gesamtzahl der Personen, wird durch Klicken auf den Spinner die Maximum und Minimum Werte der Spinner reguliert. Dieses Verhalten ist in Listing 4.3 dargestellt.

```
spnrN = new Spinner(groupParameter, SWT.BORDER);
spnrN.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(final SelectionEvent e) {
        spnrT.setMaximum(spnrN.getSelection());
        spnrT.setSelection(spnrT.getSelection());
    }
});
spnrN.setMinimum(2);
spnrN.setMaximum(500);

spnrT = new Spinner(groupParameter, SWT.BORDER);
spnrT.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(final SelectionEvent e) {
        if (spnrT.getSelection() >= spnrN.getSelection()) {
            spnrT.setMaximum(spnrT.getSelection() + 1);
            spnrN.setSelection(spnrT.getSelection());
        }
    }
});
spnrT.setMinimum(2);
spnrT.setMaximum(2);
```

Listing 4.3: Implementierung der Spinner

Für das Modul- und Secret-Textfield wurden die aus Abschnitt 2.3 bekannten `verifyListener` verwendet. Die Überprüfung dieser beiden Felder wird durch Klicken des **Select** Buttons initiiert. Es öffnet sich der **Koeffizienten-Auswahldialog**, in dem man die Koeffizenten des Polynoms wählen kann. Zuerst wird überprüft, ob die Textfields nicht leer sind. Nur in diesem Fall wird mit der weiteren Überprüfung der Parameter fortgefahren.

Es wird geprüft, ob der Wert in dem Modul-Textfield eine Primzahl ist und das Ergebnis in der Variable `isPrime` gespeichert. Falls der Wert keine Primzahl ist, wird eine Instanz der Klasse `PrimeDialog` erzeugt und aufgerufen, dabei wird im Dialog der Wert in der Farbe **rot** dargestellt. Die `PrimeDialog` Klasse beinhaltet ein `StyledText`, um die Eingabewerte farbig hervorzuheben und die zwei Buttons, **Generate next prime** und **Verify input**. Durch Drücken des **Generate next prime** Buttons wird die nächst größere Primzahl, ausgehend von dem Wert des Benutzers, berechnet. Dabei wird die Java Methode `nextProbablePrime()` für `BigInteger` verwendet, die die nächste Primzahl zurückgibt. Diese Primzahl wird in der Farbe **grün** ausgegeben, der **Verify input** Button deaktiviert und der **Ok** Button, der zuvor noch inaktiv war,

aktiviert. Man kann sich weitere Primzahlen generieren lassen, indem man weiterhin den Button **Generate next prime** drückt. Falls der Benutzer einen eigenen Wert eingeben möchte, muss er die Eingabe durch das Programm verifizieren lassen. Durch Drücken des **Verify input** Buttons wird die Eingabe überprüft. Falls es sich um eine Primzahl handelt, wird der Eingabewert in der Frabe **grün** dargestellt und der **Ok** Button zum Verlassen des Dialogs aktiviert und der **Verify input** Button deaktiviert, andernfalls wird der Eingabewert in der Farbe **rot** angezeigt. Für das farbige Hervorheben des Eingabefeldes wird ein `ExtendedModifyListener` verwendet. Sobald das Textfield verändert wird, wird die Methode `modifyText(final ExtendedModifyEvent event)` aufgerufen. In dieser Methode wird der Eingabewert in die Farbe **schwarz** gefärbt und der **Verify input** Button deaktiviert. Desweiteren wird der **Ok** Button ebenfalls deaktiviert, da das Eingabefeld modifiziert wurde und erst verifiziert werden muss. Der Quelltext des `ExtendedModyfyListener` ist in Listing 4.4 abgebildet.

```
1  private StyledText primeText;
2  primeText.addExtendedModifyListener(new ExtendedModifyListener() {
3     public void modifyText(final ExtendedModifyEvent event) {
4        style = new StyleRange();
5        style.start = 0;
6        style.length = primeText.getText().length();
7        style.foreground = BLACK;
8        style.fontStyle = SWT.BOLD;
9        primeText.setStyleRange(style);
10
11       verifyInputButton.setEnabled(true);
12       if (okButton != null) {
13          okButton.setEnabled(false);
14       }
15    }
16 });
```

Listing 4.4: Implementierung des ExtendedModifyListener

Falls der `PrimeDialog` nicht mit dem **Cancel** Button verlassen wurde, gibt die Methode `primeDialog.open()` als Rückgabewert die 0 zurück. In diesem Fall muss der Eingabewert, der in dem Dialog geändert wurde auch in der View angepasst werden. Dafür muss der `verifyListener` des Modul-Textfields entfernt werden, da man sonst keine Werte dem Textfield zuweisen kann. Nun kann der Wert übernommen und der `verifyListener` wieder hinzugefügt werden.

Auch für die Überprüfung des Wertes für das Geheimnis, benutzen wir ein Dialog, die `SecretDialog` Klasse. Da das Geheimnis nicht größer als das Modul sein darf, wird nach dem Verlassen des `PrimeDialogs` mit dem **Ok** Button überprüft, ob diese Bedingung immernoch gilt. In dem Fall, dass das Geheimnis größer ist als das Modul, erzeugen wir eine Instanz der Klasse `SecretDialog` und öffnen mit `secretDialog.open()` den Dialog. Wie auch beim `PrimeDialog` verwenden wir einen `StyledText`, um die Werte im dem Eingabefeld farblich hervorzuheben. In dem `SecretDialog` wird der Modul mit angezeigt und der Wert des Geheimnisses wird in der Farbe **rot** dargestellt. Wir benutzen eine `ExtendedModifyListener`, um die Farbe des Eingabefeldes bei Modifizierung **schwarz** zu färben. Der Button **Check input** prüft dabei lediglich, ob das Geheimnis kleiner als das Modul ist und aktiviert in diesem Fall den **Ok** Button.

Da nun die Eingabewerte für das Modul und das Geheimnis gültig sind, öffnet sich der **Koeffizienten-Auswahldialog**. Dazu wird eine Instanz der Klasse `CoefficientDialog` erzeugt und mit der Anweisung `cdialog.open()` geöffnet. Für den Aufbau des `CoefficientDialogs` werden `Spinner` verwendet. Dabei setzen wir die untere Grenze auf 0. Die obere Grenze wird durch das Modul begrenzt und beträgt dabei den Wert $Modul - 1$. Somit sind nur positive Zahlen innerhalb diesen beiden Grenzen zugelassen. Das Geheimnis wird als Koeffizient a_0 oben dargestellt.

Um sich zufällige Werte für die Koeffizienten generieren zu lassen, kann man den Button **Generate coefficients** benutzen. Dies bewirkt, dass eine Instanz der Klasse `SecureRandom` erzeugt wird, mit der man Pseudozufallszahlen generieren kann. Diese Pseudozufallszahlen werden modulo dem Modul reduziert und in die `Spinner` eingetragen. Nach Schliessen des Dialogs mit dem **Ok** Button, wird das Polynom als Formel in einem Textfeld dargestellt. Dabei wird die Methode `createPolynomialString(BigInteger[] coefficients)` aufgerufen, die aus einem `BigInteger`-Array ein Polynom-String erzeugt und zurück gibt.

Der Button **Compute shares** wird zur Berechnung der Shares aktiviert und alle anderen Einstellmöglichkeiten werden deaktiviert, um die Parameter nicht mehr zu ändern.
Durch Drücken des **Compute shares** Buttons wird eine Instanz der Klasse `ShamirsSecretSharing` erzeugt. Mit der Methode `setCoefficient(BigInteger[] coefficients)` werden die zuvor ausgewählten oder generierten Koeffizienten dem Algorithmus übergeben und mit `computeShares(Point[] shares)` die Shares berechnet. In einem Array wird das Ergebnis der Berechnung für den weiteren Ablauf zurückgegeben. Das Polynom wird im **Graph** Bereich graphisch dargestellt.

4.3.2 Das Graphical Composite

Der **Graphical** Modus ist in Abbildung 4.1 dargestellt. In Tabelle 4.1 werden die einzelnen Elemente des User Interfaces beschrieben. Die Klasse `ShamirsCompositeGraphical` ist in drei Bereiche unterteilt, den **Header**, in dem Informationen zum Algorithmus stehen, den **Graph** Bereich, in dem das Polynom dargestellt wird und dem **Settings** Bereich, der für die Parameter und Darstellung der Shares und Teilpolynome für die Rekonstruktion dient. Dabei wurde die Impementierung der Parameter bereits in Abschnitt 4.3.1 beschrieben.

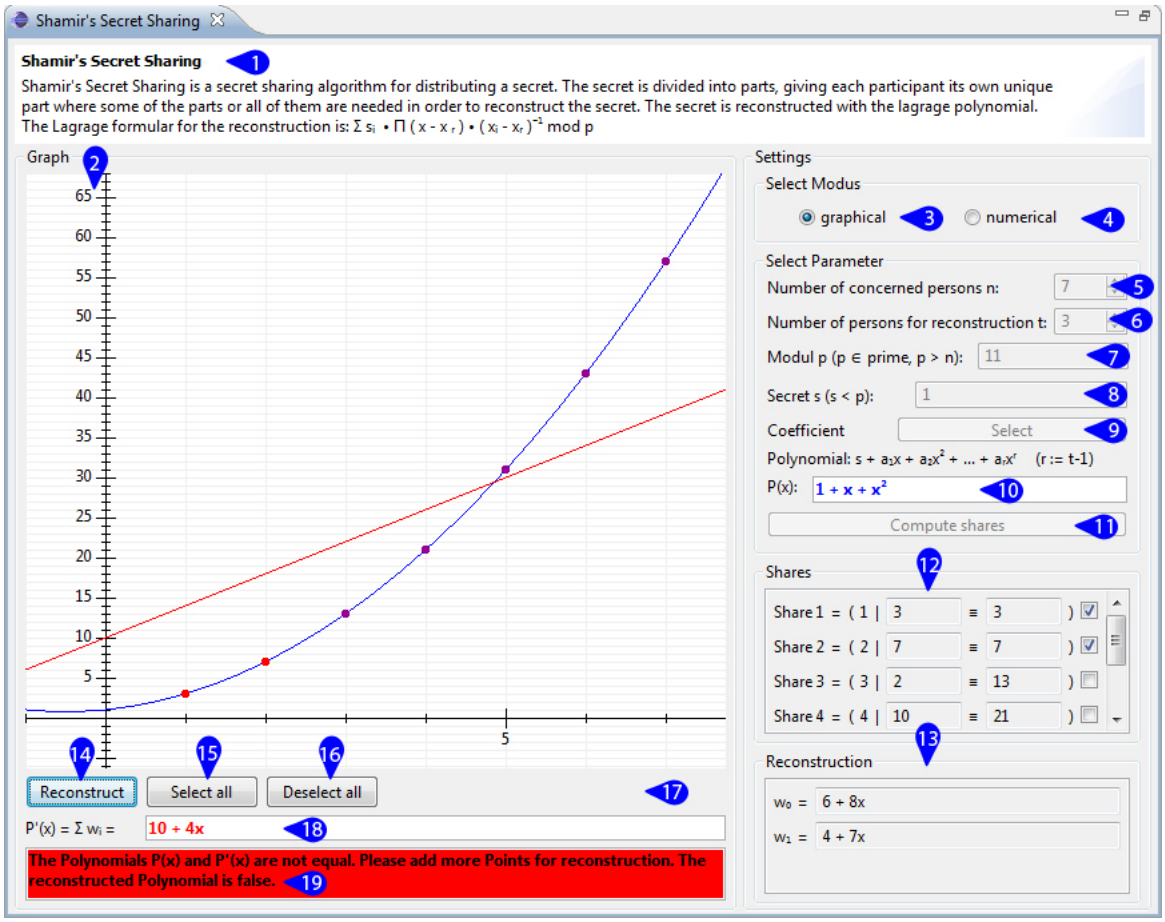

Abbildung 4.1: Das User Interface des Shamir's Secret Sharing Plug-ins im Graphical Modus

In dem **Settings** Bereich sind unten nochmals zwei Gruppen für die Darstellung der Shares und der Teilpolynome, die bei der Rekonstruktion erzeugt werden, vorhanden. In der **Shares** Gruppe werden die berechneten Shares dargestellt. Dabei werden die Shares einmal mit und ohne die Moduloreduzierung angezeigt. Die Darstellung ohne die Moduloreduzierung ist für die Zeichnung des Polynoms wichtig, da durch die Moduloreduzierung das Ergebnis der Rekonstruktion nicht dem Ausgangspolynom entspricht. Deshalb wird für die Rekonstruktion im **Graphical** Modus mit den nicht moduloreduzierten Werten weitergerechnet. Jeden Share kann man mit einer Checkbox seperat aus- bzw. abwählen. Die markierten Shares werden im **Graph** Bereich als Punkte in der Farbe **rot** auf dem Polynom dargestellt, dabei wird der nicht moduloreduzierte Wert auf dem Polynom angezeigt. In dem Plug-in haben wir den trivialen Fall, dass man mit nur

#	Typ	Initialer Zustand	Text	Beschreibung
1	Textfield	aktiviert	siehe Abbildung 4.1	Informationen zum Algorithmus.
2	Canvas	aktiviert	leer	Darstellung des Polynoms mit den berechneten Shares. Das Polynom wird in der Farbe **blau** und die Shares in der Farbe **magenta** dargestellt. Das rekonstruierte Polynom wird bei erfolgreicher Rekonstruktion in der Farbe **grün** dargestellt. Andernfalls in der Farbe **rot**. Markierte Shares werden ebenfalls in der Farbe **rot** dargestellt.
3	Radio-Button	aktiviert	graphical	Wechsel zur **Graphical** Ansicht.
4	Radio-Button	deaktiviert	numerical	Wechsel zur **Numerical** Ansicht.
5	Spinner	aktiviert	2	Festlegung der Anzahl der beteiligten Personen.
6	Spinner	aktiviert	2	Festlegung der Anzahl der nötigen Personen für die Rekonstruktion.
7	Textfield	aktiviert	leer	Eingabe des Moduls **p**.
8	Textfield	aktiviert	leer	Eingabe des Geheimnisses **s**.
9	Button	aktiviert	Select	Aufruf des Koeffizienten-Auswahl Dialogs.
10	Textfield	aktiviert	leer	Darstellung des Polynoms als Formel.
11	Button	deaktiviert	leer	Berechnung der Shares.
12	Group	deaktiviert	leer	Ausgabe der Shares.
13	Group	deaktiviert	leer	Ausgabe der Teilpolynome der Rekonstruktion.
14	Button	deaktiviert	Reconstruct	Ausführung der Rekonstruktion.
15	Button	deaktiviert	Select all	Auswahl aller Shares für die Rekonstruktion.
16	Button	deaktiviert	Deselect all	Abwahl aller Shares für die Rekonstruktion.
17	Textfield	deaktiviert	leer	Der Ausgewählte Punkt wird eingeblendet, wenn man mit dem Mauszeiger über den Share fährt.
18	Textfield	deaktiviert	leer	Darstellung des rekonstruierten Polynoms als Formel.
19	Textfield	deaktiviert	leer	Information über die Rekonstruktion, die farbig hervorgehoben wird.

Tabelle 4.1: Verhaltensbeschreibung des Shamir's Secret Sharing Plug-ins im Graphical Modus

einem Share das Geheimnis rekonstruieren kann, ausgeschlossen. Deshalb müssen mindestens zwei der Shares ausgewählt werden, damit der **Reconstruct** Button aktiviert wird und man die Berechnung durchführen kann. Zum schnellen Aus- bzw. Abwählen der Shares gibt es die Buttons **Select all** und **Deselect all**.

Der **Graph** Bereich besteht aus einer `Group`, in der wiederum ein `Composite` Objekt enthalten ist. Um nun in dem `Composite` Objekt zeichnen zu können, muss man einen `PaintListener` hinzufügen und die darin enthaltene Methode `paintControl(PaintEvent e)` überschreiben. Dazu haben wir eine Methode `drawPolynomial(e)` geschrieben, die das `PaintEvent e` übergeben bekommt. Diese Methode ist für das Zeichnen des Polynoms zuständig.

In der Methode `drawPolynomial(e)` wird der **Graphic Context** in einer Variable mit der Anweisung `GC gc = e.gc` gespeichert. Nun ist es möglich mit diesem Objekt einfache Graphikprimitive zu zeichnen. Zuerst wird das Koordinatensystem mit den Koordinatenachsen gezeichnet. Danach werden die Markierungen der Koordinatenachen geschrieben.

Für die Zeichung des Polynoms benutzen wir Transformationen. Es wird eine neue **Graphic Context** Instanz erzeugt. In einer Schleife, die über `floats` iteriert, beschreiben wir einen Pfad. Dabei wird für jeden x-Wert mit der Methode `valueAt(x)` der dazugehörige y-Wert des Polynoms beschrieben. Diese Punkte werden dann zu dem Pfad hinzugefügt und in die entsprechende Position transliert und skaliert. Der Quellcode für das Zeichnen des Polynoms ist in Listing 4.5 dargestellt.

```
1  GC polynomial = new GC(canvasCurve);
2  Path polynomPath = new Path(null);
3  float dx = 2.0f / gridSizeY;
4  polynomPath.moveTo(-10, valueAt(-10));
5
6  for (float x = -10.0f; x < size.x / 2; x += dx) {
7     polynomPath.lineTo(x, valueAt(x));
8  }
9  polynomial.setForeground(BLUE);
10
11 Transform polynomTransform = new Transform(null);
12 polynomTransform.translate(xAxisGap, yAxisGap);
13 polynomTransform.scale(gridSizeX, -gridSizeY);
14 polynomial.setTransform(polynomTransform);
15
16 polynomial.drawPath(polynomPath);
```

Listing 4.5: Plotten des Polynoms

Für die Shares, die als Punkte auf dem Polynom dargestellt werden, wird ebenfalls eine neue **Graphic Context** Instanz erzeugt und mit der primitive `fillOval()` gezeichnet.

Falls nun mehr als zwei Shares ausgewählt sind und der Button **Reconstruct** gedrückt wurde, wird zusätzlich in einem dritten **Graphic Context** das durch die ausgewählten Shares rekonstruierte Polynom dargestellt. Dabei wird die Farbe **rot**, falls die Rekonstruktion nicht erfolgreich war gewählt und die Farbe **grün** bei erfolgreicher Rekonstruktion. Zusätzlich werden in der **Reconstruction** Gruppe, die Teilpolynome dargestellt. Im **Graph**-Bereich wird das rekonstruierte Polynom als Formel angezeigt und darunter eine Information farbig hervorgehoben, ob die Rekonstruktion erfolgreich war.

Mit dem Mauszeiger kann man zu jeder Zeit über einen Share fahren, um sich den aktuell ausgewählten Punkt anzeigen zu lassen. Dabei wird der ausgewählte Punkt unter der Zeichenfläche angezeigt. Um die Position des Mauszeigers zu ermitteln, benutzen wir ein `MouseMoveListener()`, in dem wir die Methode `public void mouseMove(final MouseEvent e)` überschreiben. Durch das `MouseEvent e` Objekt können wir die aktuelle Position des Mauszeigers abfragen und übergeben diese an die Methode `nearSharePoint(Point[] sharePoints, int x, int y)`. Diese Methode liefert uns den nächsten Punkt zur aktuellen Position des Mauszeigers. Die Methode bekommt als Parameter zusätzlich zu den x- und y-Koordinaten des aktuellen Mauszeigers noch ein `sharePoints` Array, in dem sich die Shares befinden, übergeben. Den Quellcode der Methode kann man in Listing 4.6 entnehmen.

```java
private Point nearSharePoint(Point[] sharePoints, int x, int y) {
    Point point = null;
    for (int i = 0; i < sharePoints.length; i++) {
        int tmpX = sharePoints[i].getX().intValue() * gridSizeX + xAxisGap;
        int tmpY = yAxisGap - sharePoints[i].getY().intValue() * gridSizeY;

        if ((tmpX - 3 == x || tmpX - 2 == x || tmpX - 1 == x || tmpX == x
                || tmpX + 1 == x || tmpX + 2 == x || tmpX + 3 == x)
                && (tmpY - 3 == y || tmpY - 2 == y || tmpY - 1 == y
                        || tmpY == y || tmpY + 1 == y || tmpY + 2 == y || tmpY + 3 == y)) {
            point = sharePoints[i];
            return point;
        }
    }
    return point;
}
```

Listing 4.6: Finden des nächsten Punktes zur Position des Mauszeigers

4.3.3 Das Numerical Composite

Den `Numerical` Modus kann man durch Auswahl des Radio-Buttons **numerical** in der Gruppe **Select Mode** aufrufen. Die schematische Darstellung des **Numerical** Modus ist in Abbildung 4.2 dargestellt. Die Beschreibung der einzelnen Elemente des User Interfaces stehen in der Tabelle 4.2. Da die Implementierung der Parmeter bereits in Abschnitt 4.3.1 beschrieben wurde, wird an dieser Stelle nicht nochmal darauf eingegangen, da sie identisch mit dem **Graphical** Modus ist.

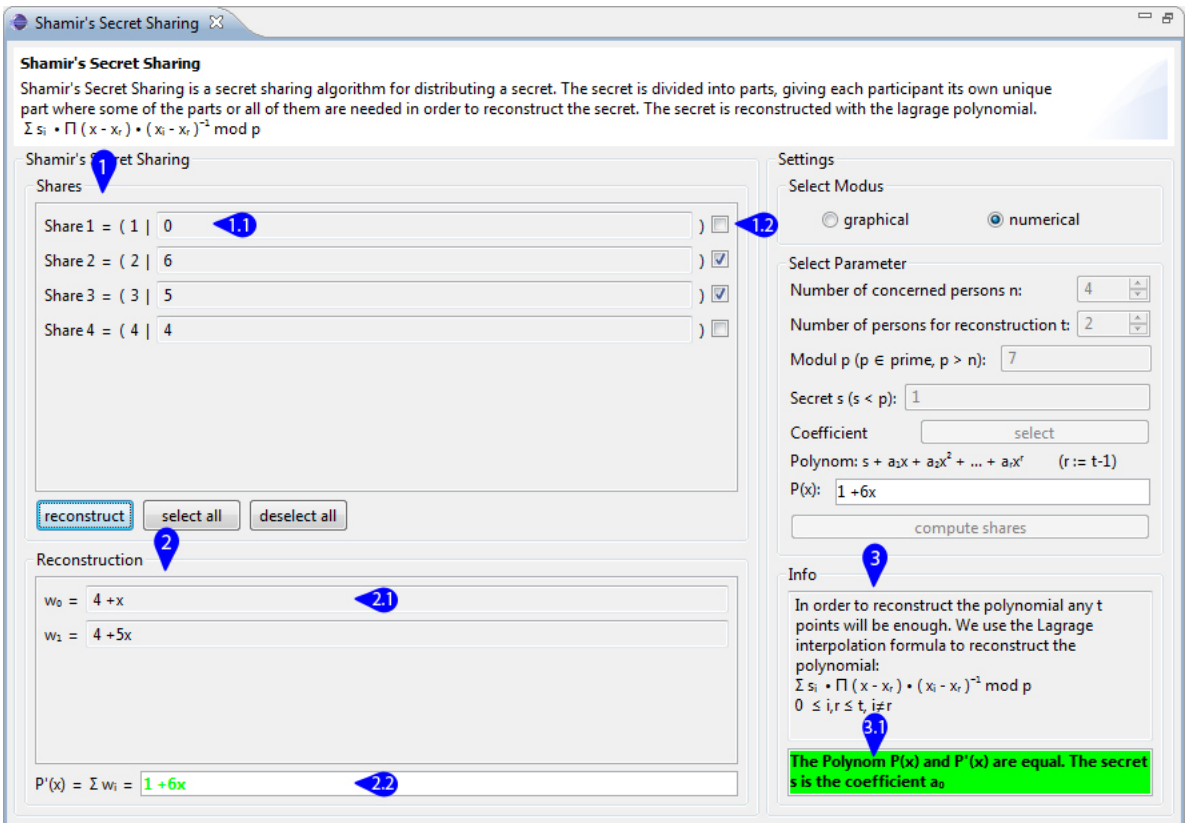

Abbildung 4.2: Das User Interface des Shamir's Secret Sharing Plug-ins im Numerical Modus

#	Typ	Initialer Zustand	Text	Beschreibung
1	Group	deaktiviert	Shares	Darstellung der Shares-Gruppe
1.1	Textfield	deaktiviert	leer	Darstellung des **y**-Wertes des Shares
1.2	Radio-Button	deaktiviert	leer	Auswahl der Shares, die für die Rekonstruktion verwendet werden sollen
2	Group	deaktiviert	Reconstruction	Darstellung der Rekonstruktion-Gruppe
2.1	Textfield	deaktiviert	leer	Darstellung der Teilpolynome
2.2	Textfield	deaktiviert	leer	Darstellung des rekonstruierten Polynoms als Formel
3	Group	deaktiviert	Info	Darstellung der Info-Gruppe
3.1	Textfield	deaktiviert	leer	Informationen darüber, ob die Rekonstruktion geglückt ist

Tabelle 4.2: Verhaltensbeschreibung des Shamir's Secret Sharing Plug-ins im Numerical Modus

Die Klasse `ShamirsCompositeNumerical` erbt von der Klasse `Composite` und implementiert das `Constants` Interface. Der Aufbau ähnelt der Klasse `ShamirsCompositeGraphical`. Die **Graph** Gruppe wurde durch eine `Composite` Gruppe, die bei der Durchführung des Algorithmus die Shares darstellt, ersetzt.

Darunter befindet sich die Gruppe **Reconstruction** für die Darstellung der Teilpolynome. Auf der rechten Seite ist eine weitere Gruppe mit dem Titel **Info** enthalten, die Informationen über die Rekonstruktion des Polynoms liefert und die **Lagrange** Formel anzeigt.

Nachdem die Parameter für die Berechnung festgelegt sind und der Button **Compute shares** gedrückt wurde, werden in der Gruppe **Shares** die berechneten Shares angezeigt. Man kann jeden Share mit einer Checkbox aus- und abwählen. Auch in diesem Modus haben wir den trivialen Fall, dass die Rekonstruktion mit nur einem Share nicht möglich ist, vernachlässigt. Es müssen mindestens zwei Shares ausgewählt sein, damit der **Reconstruct** Button aktiviert wird. Durch Klick auf den **Reconstruct** Button werden die markierten Shares für die Rekonstruktion des Polynoms verwendet und in der **Reconstruction** Gruppe werden die Teilpolynome w_is (Gleichung 4.2) angezeigt. Das rekonstruierte Polynom wird farbig hervorgehoben und eine Information in der **Info** Gruppe angezeigt. Dabei wird die Farbe **rot** gewählt, falls die Rekonstruktion nicht erfolgreich war und die Farbe **grün** bei erfolgreicher Rekonstruktion.

4.4 Die Funktionsweise des Plug-ins

Das **Shamir's Secret Sharing** Plug-in läßt sich über das Menü **Visualisierungen** oder über die View **Algorithmen** im Tab **Visualisierungen** starten und in seiner Ansicht maximieren (Abbildung 4.3).

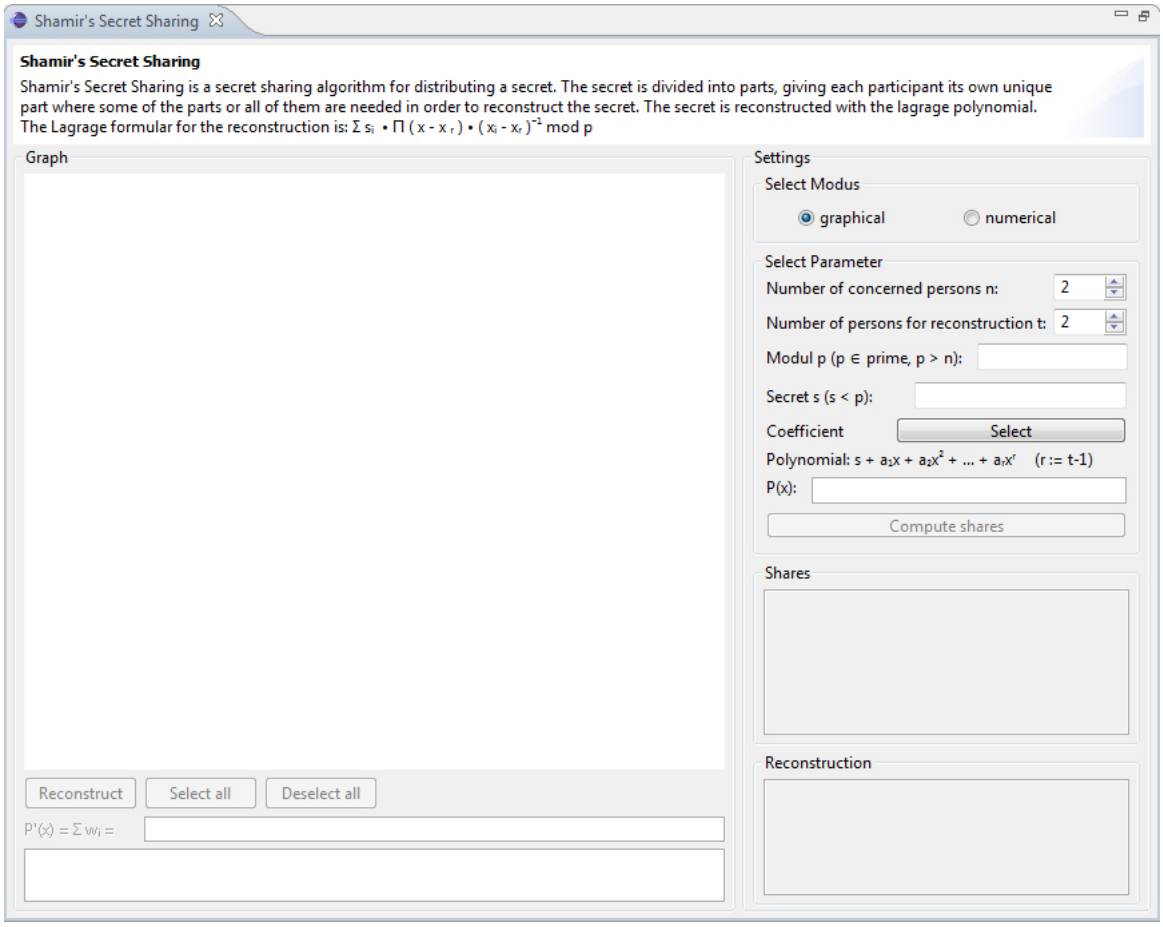

Abbildung 4.3: Das Shamir's Secret Sharing Plug-in

Das Plug-in gliedert sich in drei Bereiche: Den **Header**, den **Grafik-** oder **Numerik-**Bereich und den **Settings** Bereich. Im **Header** ist eine kurze Beschreibung über das Plug-in angezeigt. In dem **Grafik-** bzw. **Numerik-**Bereich wird die Visualisierung dargestellt und in dem **Settings** Bereich werden die Parameter für den Ablauf festgelegt. Zuerst entscheidet der Benutzer, ob er eine **graphische-** oder **numerische-**Darstellung des Plug-ins möchte. Dazu kann er im **Settings** Bereich unter dem Punkt **Select Modus** den Modus einstellen (Abbildung 4.4).

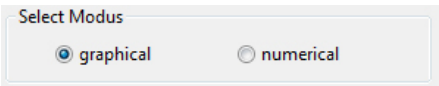

Abbildung 4.4: Auswahl des Modi

Je nachdem für welchen Modus der Benutzer sich entscheidet, ändert sich, bis auf die im **Settings** Bereich einstellbaren Parameter, die Ansicht der View. In Abbildung 4.5 ist der **Numerical** Modus dargestellt.

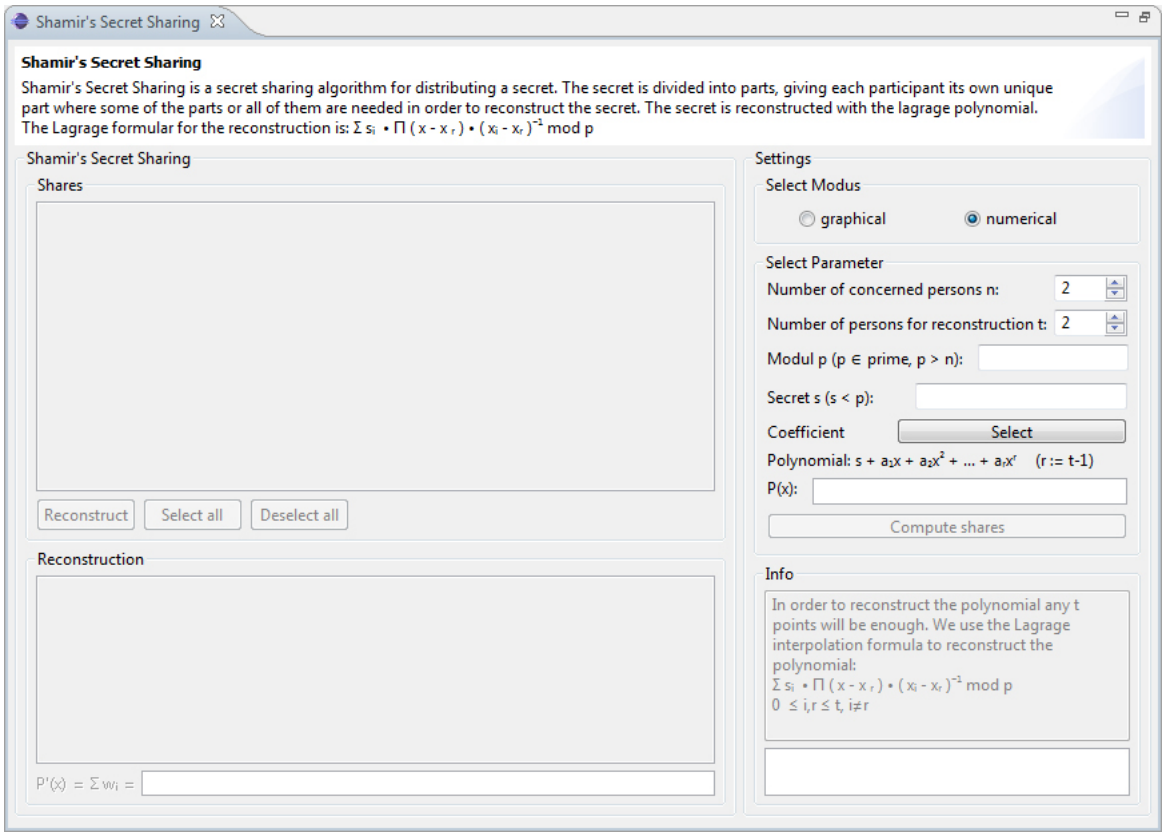

Abbildung 4.5: Der Numerical Modus

4.4.1 Festlegung der Parameter für die Berechnung

Unabhängig davon welcher Modus ausgewählt ist, kann man in dem Bereich **Select Parameter** die nötigen Parameter für den Ablauf des Algorithmus einstellen. Man wählt die Anzahl der Personen aus, an den die **Shares** verteilt werden sollen und die Anzahl der Personen, die für die Rekonstruktion nötig sind. Desweiteren wählt man eine Primzahl **p** als Modul und ein Geheimnis **s**, das kleiner als das Modul ist für die Berechnung aus (Abbildung 4.6).

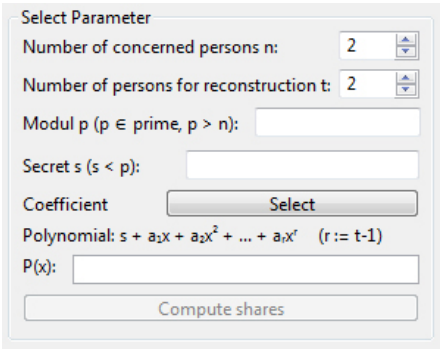

Abbildung 4.6: Einstellen der Parameter

Durch Drücken des **Select** Buttons, öffnet sich ein Dialogfeld, in dem man die Koeffizienten des Polynoms auswählen kann. Dabei ist der Koeffizient a_0 das gewählte Geheimnis und kann nicht verändert werden. Zulässige Werte für die Koeffizienten sind nur positive Zahlen, von Null bis zu der oberen Grenze $p - 1$. Es ist auch möglich durch Betätigen des Buttons **Generate coefficients** sich gültige Werte generieren zu lassen (Abbildung 4.7).

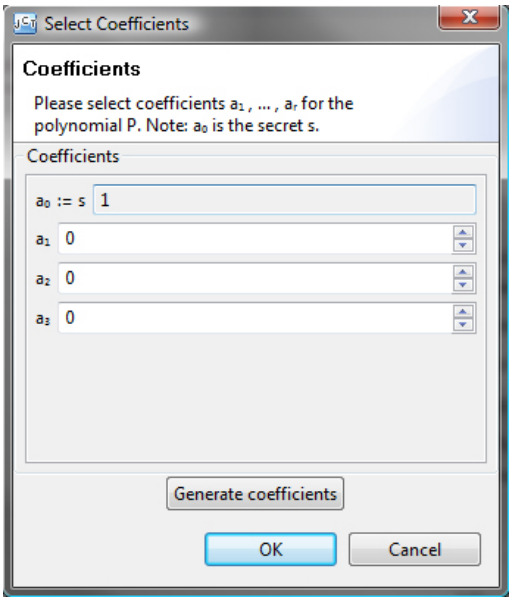

Abbildung 4.7: Wählen der Koeffizienten

Durch Drücken des **Ok** Buttons wird das Dialogfeld geschlossen und das Polynom als Formel dargestellt. Falls der Wert für den Modul Parameter **p** keine Primzahl ist, öffnet sich ein Dialogfeld, in dem man die Eingabe korrigieren kann (Abbildung 4.8). Dabei wird der ungültige Wert in der Farbe **rot** dargestellt. Es gibt nun die Möglichkeit sich eine Primzahl, ausgehend von der Zahl die eingegeben wurde, generieren zu lassen oder der Benutzer kann eine andere Zahl eingeben. Falls man sich eine Zahl durch das Programm generieren läßt, wird der Button **Verify input** deaktiviert, da der vorgeschlagene Wert gültig und die Verifikation nicht nötig ist. Es können nun weitere Zahlen durch Drücken des **Generate next prime** Buttons berechnet werden. Falls der Benutzer eine Eingabe tätigt, wird die Zahl zuerst in der Farbe **schwarz** dargestellt. Der Benutzer muss nun die eingegebene Zahl durch das Programm verifizieren lassen. Dabei drückt er den **Verify input** Button. Falls es sich um eine Primzahl handelt, wird die Zahl in **grün** dargestellt und der **Ok** Button aktiviert, andernfalls wird die Zahl **rot** gefärbt und der Benutzer muss entweder eine andere Zahl eingeben oder sich vom Programm eine gültige Primzahl generieren lassen. Sobald eine gültige Primzahl eingegeben wurde, kann der Benutzer den Dialog mit dem **Ok** Button verlassen.

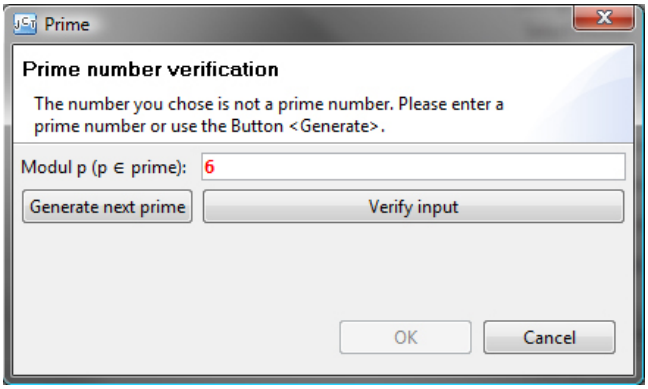

Abbildung 4.8: Korrektur des Moduls, falls Eingabewert keine Primzahl ist

Nun kann es sein, dass das Geheimnis **s** entweder schon vor dem Verifikationsdialog kleiner als das Modul **p** war oder währenddessen durch die Eingabe des Benutzers kleiner als das Modul geworden ist. Für diesen Fall öffnet sich ein weiterer Dialog, um die Bedingung, dass das Geheimnis **s** echt kleiner als das Modul **p** sein muss, zu erfüllen. Der Benutzer kann nun das Geheimnis **s** ändern, indem er eine kleinere Zahl als das Modul **p** eingibt. Zur besseren Übersicht wird das Modul oben angezeigt. Die Eingabe des Benutzers muss verifiziert werden, dazu wird der **Check input** Button gedrückt. Wenn der Wert für das Geheimnis **s** echt kleiner als das Modul **p** ist, wird der Wert in **grün** dargestellt, andernfalls in der

Farbe **rot**. Falls es sich um einen gültigen Wert handelt, wird der **Ok** Button aktiviert und der Benutzer kann den Dialog mit dem **Ok** Button verlassen (Abbildung 4.9). Das Polynom wird als Formel dargestellt.

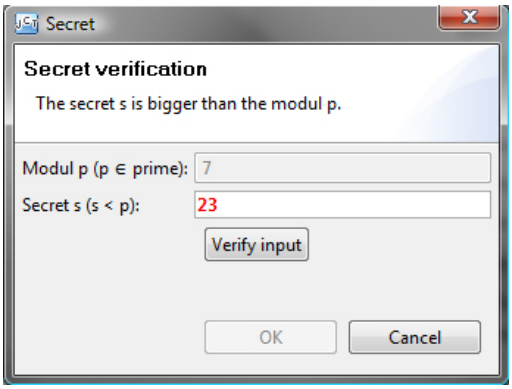

Abbildung 4.9: Korrektur des Geheimnisses, falls Geheimnis echt größer als das Modul ist

Nun kann man mit Betätigen des Buttons **Compute shares** die Shares berechnen, die an die Personen verteilt werden (Abbildung 4.10).

4.4.2 Der Grafik Modus

Die berechneten Shares werden in dem Bereich **Shares** angezeigt (Abbildung 4.11). Auf der linken Seite wird der dazugehörige Graph des Polynoms im **Graph** Bereich dargestellt. Die einzelnen Shares werden im Graphen als Punkte in der Farbe **magenta** angezeigt (Abbildung 4.12). In dem **Shares** Bereich kann man die Shares, die man für die Rekonstruktion verwenden möchte, per Checkbox auswählen. Dabei werden die augewählten Shares zur besseren Übersicht im Graphen in der Farbe **rot** dargestellt. Es müssen mindestens zwei Shares ausgewählt werden, damit der Button **Reconstruct** aktiviert wird. Um alle Shares aus- oder abzuwählen, kann man auch die Buttons **Select all** oder **Deselect all** benutzen.

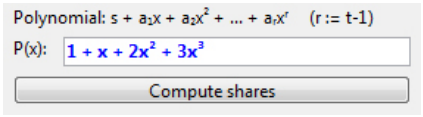

Abbildung 4.10: Darstellung des Polynoms als Formel und Button für die Shareberechnung

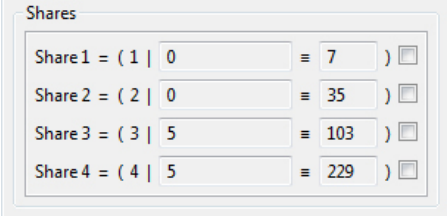

Abbildung 4.11: Darstellung der berechneten Shares im Grahical Modus

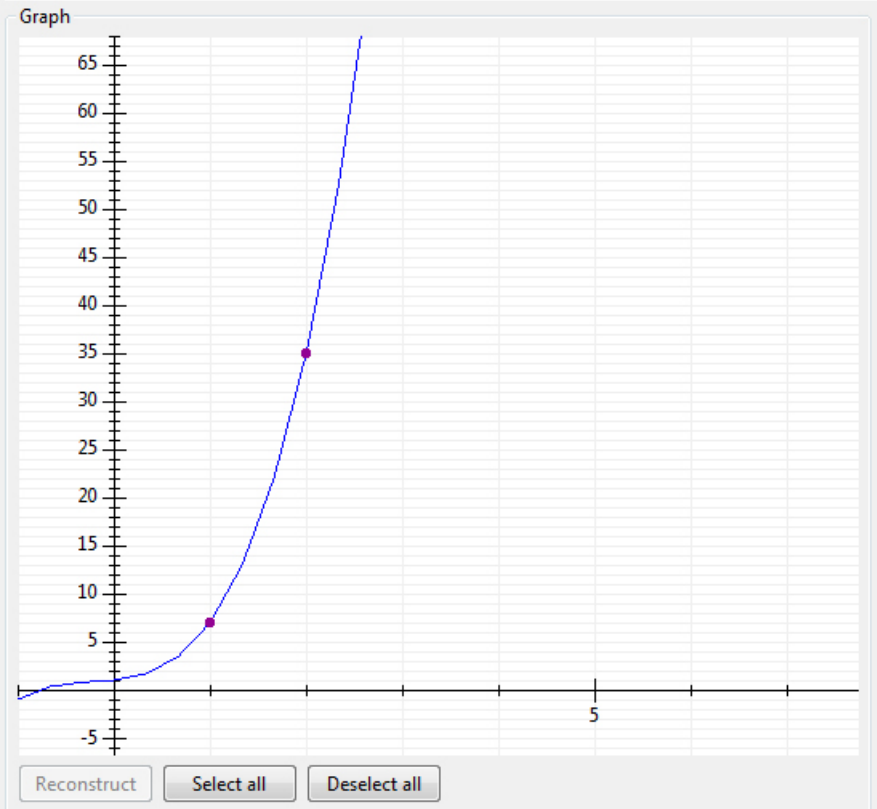

Abbildung 4.12: Darstellung des Polynoms als Graph

Nach dieser Auswahl wird durch den Button **Reconstruct** die Berechnung der Rekonstruktion des Polynoms durchgeführt. Im Bereich **Reconstruction** werden die Zwischenergebnisse der Berechnung angezeigt (Abbildung 4.13). Im unteren Teil des **Graph** Bereichs wird das rekonstruierte Polynom als Formel angezeigt. Dabei wird das Ergebnis in der Farbe **grün** angezeigt, falls das Polynom vollständig rekonstruiert wurde und die Farbe **rot** gewählt, falls nicht genug Punkte für die Rekonstruktion ausgewählt wurden (Abbildung 4.14). Für den Fall, dass das Polynom nicht richtig rekonstruiert wurde, werden beide Polynome als Graph in unterschiedlichen Farben dargestellt (Abbildung 4.15).

Nun kann man mit dem Mauszeiger über die einzelnen Punkte fahren, um sich die Punkte anzeigen zu lassen. Dabei wird der Punkt unter der Grafikfläche eingeblendet (Abbildung 4.16).

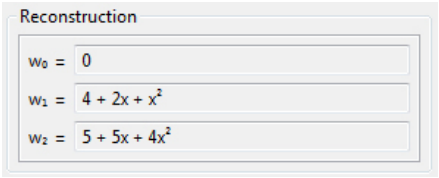

Abbildung 4.13: Darstellung der Zwischenergebnisse

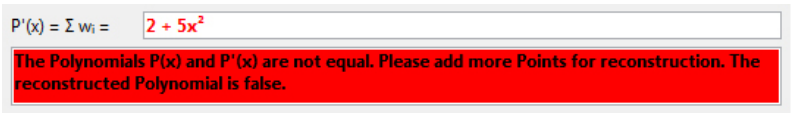

Abbildung 4.14: Darstellung des Endergebnisses

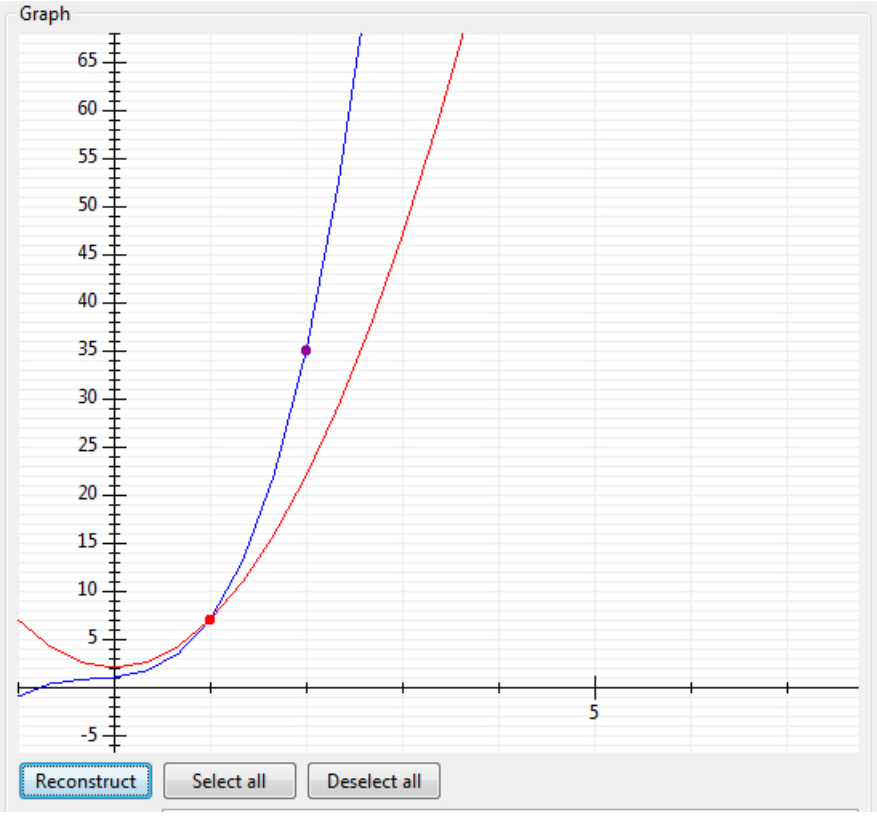

Abbildung 4.15: Darstellung des Endergebnisses als Graph bei nicht erfolgreicher Rekonstruktion

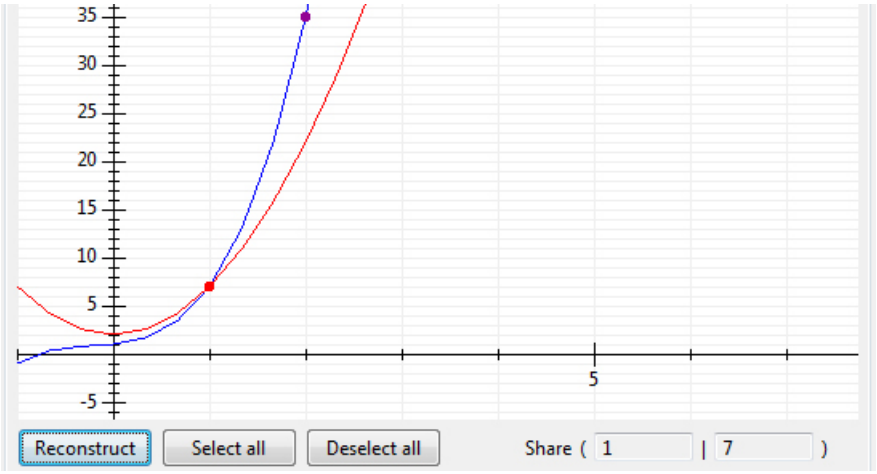

Abbildung 4.16: Mausover Effekt des Graphen

4.4.3 Der Numerik Modus

Der **Numerik** Modus eignet sich für große Zahlen. Dabei ist der Bereich **Select Parameter** identisch mit dem **Graphik** Modus und kann in Abschnitt 4.4.1 nachgelesen werden. Auf der rechten Seite unten befindet sich der **Info** Bereich, in dem die Berechnungsvorschrift und das jeweilige Endergebnis nach der Berechnung angezeigt wird (Abbildung 4.17).

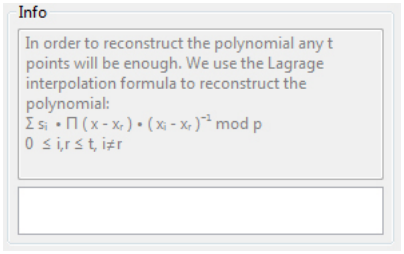

Abbildung 4.17: Info Bereich mit der Berechnungsvorschrift

Die linke Seite wird in zwei zusätzliche Bereiche unterteilt, den **Shares** Bereich und den **Reconstruction** Bereich. In dem **Shares** Bereich werden die berechneten Shares angezeigt. Der Bereich ist für große Werte gut geeignet (Abbildung 4.18). Auch hier kann man per Checkbox die jeweiligen Shares für die Rekonstruktion auswählen. Um alle aus- bzw abzuwählen, kann man die Buttons **Select all** bzw. **Deselect all** benutzen. Wiederum muss man mindestens zwei Shares auswählen, damit man den **Reconstruct** Button betätigen kann.

Abbildung 4.18: Shares Bereich im Numerik-Modus

Danach werden die Zwischenergebnisse in dem Bereich **Reconstruction** angezeigt. Das rekonstruierte Polynom wird darunter in der Farbe **grün**, falls die Rekonstruktion erfolgreich war und in der Farbe **rot**, falls zu wenige Shares ausgewählt wurden, angezeigt (Abbildung 4.19). Desweiteren wird zusätzlich im **Info** Bereich eine Information zum Endergebnis ausgegeben (Abbildung 4.20).

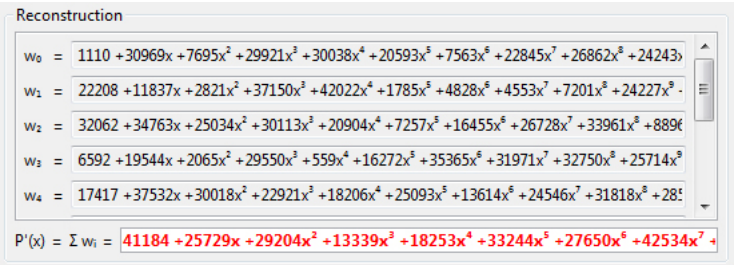

Abbildung 4.19: Reconstruction Bereich im Numerik-Modus

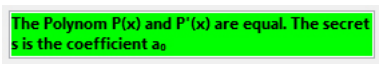

Abbildung 4.20: Information zum Ergebnis

5 Ausblick

In dieser Diplomarbeit wurden die drei Plug-ins **Extended Euclidean**, **Chinese Remainder Theorem** und **Shamir's Secret Sharing** für die E-Learning Plattform **JCrypTool** implementiert und vorgestellt. Dabei wurde großer Wert auf die Usability der Plug-ins gelegt. Jedem Plug-in wurde eine Kontext-Hilfe hinzugefügt, in der der Benutzer die Beschreibung des Algorithmus und die Funktionsweise des Plug-ins nachlesen kann.

Diese Diplomarbeit zeigt die Entwicklung von Plug-ins für die E-Learning Plattform **JCrypTool**. In dem Buch [Buc08], das als Grundlage für die Implementierung der Plug-ins verwendet wurde, gibt es viele weitere interessante Algorithmen, die man entwickeln kann.

Ein interessantes Plug-in wäre z.B. ein Probeklausur-Generator, in dem Aufgaben zu verschiedenen Algorithmen generiert werden. Ansätze dafür sind schon in den Plug-ins **Extended Euclidean** und **Chinese Remainder Theorem** als Export-Funktion implementiert.

Für **Shamir's Secret Sharing** Plug-in könnte man für den **Graphical** Modus eine Zoom-Funktion implementieren. Damit könnte man in die dargestellten Polynome ein- bzw. auszoomen. Dadurch erhöht man das Verständnis des Verfahrens, da bei der jetzigen Version des Plug-ins die berechneten Shares schon bei Polynomen von Grad 5 sehr groß werden und nicht mehr in der Zeichenfläche liegen.

Ein weiterer interesanter Algorithmus ist in [SSc] beschrieben. Hier wird ein **Secret Sharing** Verfahren beschrieben, das den **Chinese Remainder Theorem** benutzt. Um dieses Verfahren zu implementieren, könnte man in dem **Shamir's Secret Sharing** Plug-in eine weitere View erstellen, die diesen Algorithmus implementiert. Den Algorithmus dazu kann man so schreiben, dass das **Chinese Remainder Theorem** Plugin verwendet wird.

Für alle drei Plug-ins wäre ein Modus denkbar, in dem man schon vorher das Endergebnis eingibt und dieses Endergebnis dann durch das Plug-in verifizieren läßt, ob die Eingabe des Endergebnisses richtig ist. Bei falscher Eingabe könnte dieser Modus die Stelle anzeigen, in der es zum Konflikt beim Algorithmus gekommen ist.

Literaturverzeichnis

[Buc08] Buchmann, Johannes: *Einführung in die Kryptographie*. 4., erweiterte Auflage. Springer Verlag, 2008. – ISBN 978–3–540–74451–1

[CR07] Chris Rupp, Barbara Z. Stefan Queins Q. Stefan Queins: *UML 2 Glasklar*. 3. Auflage. Carl Hanser Verlag, 2007. – ISBN 978–3–446–41118–0

[DG03] David Gallardo, Robert M. Ed Burnette B. Ed Burnette: *Eclipse in Action: A Guide for Java Developers*. 1. Auflage. Manning Publications, 2003. – ISBN 978–1–930110–96–0

[EC09] Eric Clayberg, Dan R.: *Eclipse Plug-ins*. 3. Auflage. Addison-Wesley Verlag, 2009. – ISBN 978–0–321–55346–1

[EG09] Erich Gamma, Ralph Johnson John V. Richard Helm H. Richard Helm: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. 1. Auflage. Addison-Wesley Verlag, 2009. – ISBN 978–3–8273–2824–3

[JCt09] *Getting started with JCrypTool*. http://jcryptool.sourceforge.net/JCrypTool/Documentation_files/getting_started_with_jcryptool.pdf. Version: September 2009

[Kna04] Knappen, Jörg: *Schnell ans Ziel mit LATEX 2e*. 2. Auflage. Oldenbourg Wissenschaftsverlag, 2004. – ISBN 978–3–486–27447–3

[Kop00] Kopka, Helmut: *LaTeX Band 1: Einführung*. 3., überarbeitete Auflage. Addison-Wesley, 2000. – ISBN 978–3–8273–7038–8

[Low06] Lowagie, Bruno: *iText in Action*. 1. Auflage. Manning, 2006

[MG00] Michel Goossens, Alexander S. Frank Mittelbach M. Frank Mittelbach: *Der LaTeX-Begleiter*. 1. Auflage. Addison-Wesley Verlag, 2000. – ISBN 978–3–8273–7044–0

[RSA78] Rivest, RL ; Shamir, A. ; Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. In: *Communications of the ACM* 21 (1978), Nr. 2, 126. http://people.csail.mit.edu/rivest/Rsapaper.pdf

[Sha79] Shamir, Adi: How to share a secret. (1979). http://groups.csail.mit.edu/cis/crypto/classes/6.857/papers/secret-shamir.pdf

[SSc] *Secret Sharing using the Chinese Remainder Theorem*. http://en.wikipedia.org/wiki/Secret_Sharing_using_the_Chinese_Remainder_Theorem

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis