Entwurf und Implementierung eines Plug-Ins für JCrypTool

RSA, DSA und ElGamal Visualisierungen
Bachelor-Thesis von Michael Gaber
September 2009
Entwurf und Implementierung eines Plug-Ins für JCrypTool
RSA, DSA und ElGamal Visualisierungen

vorgelegte Bachelor-Thesis von Michael Gaber

Tag der Einreichung:
Erklärung zur Bachelor-Thesis

Hiermit versichere ich die vorliegende Bachelor-Thesis 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 27.09.2009

(Michael Gaber)
## Inhaltsverzeichnis

1 Einleitung und Motivation ................................................. 1

2 Das RSA Plug-In .......................................................... 2
   2.1 Oberfläche .................................................................. 2
   2.2 Ablauf ....................................................................... 2
      2.2.1 Ablauf der Schlüsselerzeugung ................................. 2
         2.2.1.1 Ablauf der Erzeugung eines öffentlichen Schlüssels 3
         2.2.1.2 Ablauf der Erzeugung eines privaten Schlüssels ....... 5
   2.3 Texteingabe .................................................................. 7
      2.3.1 Klartexteingabe ..................................................... 7
      2.3.2 Hashfunktion ....................................................... 7
      2.3.3 Chiffretexteingabe ................................................. 9
   2.4 Berechnungsanzeige und schnelles Exponentieren ................. 10
   2.5 Ergebnisbereich .......................................................... 10
   2.6 Optionen .................................................................... 10
   2.7 Vergleich mit CrypTool ............................................... 11

3 Das ElGamal Plug-In ......................................................... 13
   3.1 Oberfläche ................................................................. 13
   3.2 Ablauf ....................................................................... 13
      3.2.1 Ablauf der Schlüsselerzeugung ................................. 13
         3.2.1.1 Verwendung vorhandener Schlüssel ..................... 16
         3.2.1.2 Ablauf der Erzeugung eines öffentlichen Schlüssels .. 16
         3.2.1.3 Ablauf der Erzeugung eines privaten Schlüssels ....... 16
   3.3 Texteingabe .................................................................. 19
      3.3.1 Klartexteingabe ..................................................... 20
      3.3.2 Chiffretexteingabe ................................................. 20
   3.4 Parameterwahl ............................................................. 21
   3.5 Berechnungsanzeige ...................................................... 22
   3.6 Ergebnisbereich .......................................................... 22
   3.7 Optionen .................................................................... 22
   3.8 Vergleich mit CrypTool ............................................... 23

4 Das DSA Plug-In ............................................................... 24
   4.1 Oberfläche ................................................................. 24
   4.2 Ablauf ....................................................................... 24
      4.2.1 Ablauf der Schlüsselerzeugung ................................. 24
         4.2.1.1 Verwendung vorhandener Schlüssel ..................... 26
         4.2.1.2 Ablauf der Erzeugung eines öffentlichen Schlüssels .. 26
         4.2.1.3 Ablauf der Erzeugung eines privaten Schlüssels ....... 26
4.3 Texteingabe ........................................... 30
4.4 Parameterwahl .......................................... 30
4.5 Berechnungsanzeige ..................................... 30
4.6 Ergebnisbereich .......................................... 30
4.7 Optionen .................................................. 31
4.8 Vergleich mit CrypTool ................................. 31

5 Das Dummy Plug-In ................................. 32

6 Interessante Codeschnipsel ........................ 34
   6.1 Sub- und Superscript ............................... 34
   6.2 Threading .............................................. 36

7 Rück- und Ausblick ................................... 38
   7.1 Ausblick .............................................. 38
<table>
<thead>
<tr>
<th>Abschnitt</th>
<th>Seitennummer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Startaussehen des RSA-Plug-Ins</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Ablaufplan des gesamten RSA-Plug-Ins</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Screenshot der RSA-Publickeyerzeugung</td>
<td>5</td>
</tr>
<tr>
<td>2.4 Ablaufplan der RSA-Publickeyerzeugung</td>
<td>5</td>
</tr>
<tr>
<td>2.5 Screenshot der RSA-Privatekeyerzeugung</td>
<td>6</td>
</tr>
<tr>
<td>2.6 Ablaufplan der RSA-Privatekeyerzeugung</td>
<td>8</td>
</tr>
<tr>
<td>2.7 Ablauf einer Runde SHA-1</td>
<td>9</td>
</tr>
<tr>
<td>2.8 Endaussehen des RSA-Plug-Ins</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Startaussehen des ElGamal-Plug-Ins</td>
<td>14</td>
</tr>
<tr>
<td>3.2 Ablaufplan des gesamten ElGamal-Plug-Ins</td>
<td>15</td>
</tr>
<tr>
<td>3.3 Screenshot der ElGamal-Publickeyerzeugung</td>
<td>17</td>
</tr>
<tr>
<td>3.4 Ablaufplan der ElGamal-Publickeyerzeugung</td>
<td>17</td>
</tr>
<tr>
<td>3.5 Screenshot der ElGamal-Privatekeyerzeugung</td>
<td>18</td>
</tr>
<tr>
<td>3.6 Ablaufplan der ElGamal-Privatekeyerzeugung</td>
<td>19</td>
</tr>
<tr>
<td>3.7 Screenshot der Plaintexteingabe</td>
<td>20</td>
</tr>
<tr>
<td>3.8 Screenshotmontage der Parameterwahl</td>
<td>21</td>
</tr>
<tr>
<td>3.9 Endaussehen des ElGamal-Plug-Ins</td>
<td>23</td>
</tr>
<tr>
<td>4.1 Startaussehen des DSA-Plug-Ins</td>
<td>25</td>
</tr>
<tr>
<td>4.2 Ablaufplan des gesamten DSA-Plug-Ins</td>
<td>27</td>
</tr>
<tr>
<td>4.3 Screenshot der DSA-Publickeyerzeugung</td>
<td>28</td>
</tr>
<tr>
<td>4.4 Screenshot der DSA-Privatekeyerzeugung</td>
<td>29</td>
</tr>
<tr>
<td>5.1 Aussehen des Dummy Plug-Ins</td>
<td>32</td>
</tr>
<tr>
<td>5.2 Aussehen des Dummy Wizards</td>
<td>33</td>
</tr>
</tbody>
</table>
Listings

6.1 Grundangaben für Superscripts ........................................... 34
6.2 Anwendung von Superscripts ............................................. 35
6.3 Threading in SWT ............................................................. 36
6.4 Eintragungsrunnable ......................................................... 37
1 Einleitung und Motivation

Cryptool\(^1\) ist ein weltweit für Lehre und Schulung eingesetztes Tool mit dem ein Verständnis für Kryptographie erlangt werden soll. Das Tool hat bereits zahlreiche Preise wie den „European Information Security Award“ und den „IT-Sicherheitspreis NRW“ gewonnen. Außerdem wurde die Universität Siegen mit Cryptool einer der „365 Orte im Land der Ideen“\(^2\). Ein großes Problem bleibt trotz dieser offensichtlichen Qualität und weltweiten Akzeptanz\(^3\) dennoch bestehen: Cryptool ist in C++ geschrieben und infolge dessen plattformabhängig und läuft nur auf Windows Plattformen.

Um dieses Problem zu umgehen wurde das JCrypTool Projekt\(^4\) ins Leben gerufen, das einen Nachfolger zu CrypTool entwickelte, der auf der Eclipse Rich Client Platform\(^5\) basiert. Dies macht JCrypTool prinzipiell auf jeder Plattform lauffähig, für die es eine Implementierung des Standard Widget Toolkit\(^6\) gibt, was Windows, Linux, MacOS, Solaris und weitere Unix-Derivate umfasst.


Die so zur Verfügung gestellten Algorithmen bieten über eine eigene Ansicht einen angenehmen Weg direkt mit Kryptographie zu arbeiten und Dateien oder Editorfenster zu ver- oder entschlüsseln, sowie weitere kryptographische Operationen darauf anzuwenden. Leider ist jedoch FlexiProvider als Kryptographieprovider nicht dafür gedacht, seine Abläufe besonders nutzerfreundlich darzustellen, so dass Visualisierungen und Erklärungen zu den einzelnen Algorithmen bisher noch Mangelware sind.

In diesem Punkt setzt die vorliegende Bachelorarbeit an, die für die gebräuchlichen Algorithmen RSA, DSA und ElGamal Visualisierungen entwickelte, die das Verständnis für den Algorithmus vertiefen und gleichzeitig zum Spielen anregen sollen, indem sie dem Nutzer bei der Verwendung möglichst umfassende Freiheiten lassen.

---

1 Zu finden unter \url{http://cryptool.org/}
2 ebd.
3 Cryptool spricht von 3000 Downloads pro Monat.
4 vgl. \url{http://jcryptool.sourceforge.net/JCrypTool/Home.html}
5 vgl. \url{http://wiki.eclipse.org/index.php/Rich_Client_Platform}
6 Siehe \url{http://wiki.eclipse.org/SWT}
7 zu finden unter \url{http://www.bouncycastle.org/}
Das Plug-In für RSA, einen der heutigen Standardalgorithmen für asymmetrische Verschlüsselung und Signatur [8], soll dem Benutzer die verschiedenen Schritte der Ver- und Entschlüsselung mittels RSA verdeutlichen. Dazu werden ihm die Möglichkeiten gegeben, alle Parameter des Algorithmus selbst zu bestimmen und teilweise auch Parameter selbst einzugeben. Wählt der Nutzer diese Möglichkeit, so ist es ihm auch gestattet Werte einzugeben, die den Anforderungen des Verfahrens eigentlich nicht entsprechen, zum Beispiel zusammengesetzte Zahlen für p und q.

### 2.1 Oberfläche


Zu jedem Zeitpunkt ist außerdem ein Reset möglich um alle bisher eingegebenen Daten zu löschen und das Plug-In in seinen Ausgangszustand zurück zu versetzen.

### 2.2 Ablauf


#### 2.2.1 Ablauf der Schlüsselerzeugung

Nachdem der User festgelegt hat, was für eine Aktion er durchführen möchte und damit auch festgelegt hat, ob ein privater oder ein öffentlicher Schlüssel benötigt wird, kann er im nächsten Schritt einen Wizard zur Schlüsselerzeugung bzw. -auswahl starten. Die Auswahlseiten bieten jeweils ein Drop-Down-Menü, das alle passenden Schlüssel enthält, die mit Kontaktname und Schlüssellänge angezeigt werden. Für das Laden eines privaten Schlüssels ist zusätzlich die Eingabe
Abbildung 2.1: Startaussehen des RSA-Plug-Ins

 eines Kennworts notwendig, das bei der Speicherung festgelegt wurde und für dessen Eingabe ein Textfeld zur Verfügung steht.

Es ist hier auch das Laden und Verwenden der mittels FlexiProvider und der FlexiProvider-Perspektive erzeugten Schlüssel möglich, aufgrund der Größe der verwendeten Zahlen leidet die Übersichtlichkeit jedoch so stark, dass die Verwendung nicht empfohlen wird.

Die Verfahren für die Erzeugung der privaten (Abschnitt 2.2.1.1) und öffentlichen (Abschnitt 2.2.1.2) Schlüssel unterscheiden sich massiv, so dass sie in eigenen Abschnitten behandelt werden.

### 2.2.1.1 Ablauf der Erzeugung eines öffentlichen Schlüssels

Der öffentliche Schlüssel eines RSA-Schlüsselpaares besteht aus dem öffentlichen Exponenten $e$ und dem RSA-Modul $N$. Der Ablauf der Eingabe ist in Abbildung 2.4 dargestellt, während Abbildung 2.3 eine Screenshot der Seite zeigt. Diese beiden Werte werden über jeweils ein Textfeld eingegeben, das lediglich die Eingabe von Zahlen erlaubt. Ist $N$ nicht größer als 256\(^1\), so wird ein Fehler angezeigt und ein Fortsetzen ist nicht möglich. Außerdem wird $N$ darauf überprüft, ob es das Produkt zweier Primzahlen ist. Ist dies nicht der Fall, wird ein ersatzweises $N$ vorgeschlagen, das

\(^1\) Diese Beschränkung wurde Aufgrund der Konversion der Buchstaben in Zahlwerte mittels ASCII-Code getroffen.
Abbildung 2.2: Ablaufplan des gesamten RSA-Plug-Ins
Abbildung 2.3: Screenshot der RSA-Publickeyerzeugung

Abbildung 2.4: Ablaufplan der RSA-Publickeyerzeugung

in etwa der gleichen Größe wie die angegebene Zahl entspricht und eine Warnung ausgegeben. Ein Fortsetzen ist dennoch möglich.

Auch für e werden einige rudimentäre Tests durchgeführt, nämlich insbesondere, dass die Zahl größer als 1 und kleiner als N ist. Außerdem wird sichergestellt, dass e nicht größer als \( \lfloor (\sqrt{N} - 1)^2 \rfloor \) ist. Dieser letzte Test ist jedoch lediglich eine Abschätzung für ein valides e, da die Faktoren p und q natürlich nicht bekannt sind und somit auch das korrekte \( \varphi(N) \) unbekannt ist.

2.2.1.2 Ablauf der Erzeugung eines privaten Schlüssels

Abbildung 2.5: Screenshot der RSA-Privatekeyerzeugung
Sobald sowohl \( p \) als auch \( q \) gewählt wurden und unterschiedlich sind, werden \( N = p \times q \) und \( \phi(N) \) berechnet. Letztere Zahl ist die eulersche Phi-Funktion, deren Wert die Anzahl der zu \( N \) teilerfremden Zahlen zwischen 1 und \( N \) angibt. Für eine Primzahl \( x \) ist ihr Wert \( x-1 \), für zusammengesetzte Zahlen lässt sie sich über die Primfaktorzerlegung der Zahl berechnen, da sie multiplikativ ist. Für \( x = y \times z \) mit \( y \) und \( z \) Primzahlen ist

\[
\phi(x) = \phi(y) \times \phi(z) = (y - 1) \times (z - 1)
\]

Aus \( \phi(N) \) leiten sich auch direkt die möglichen Werte für den öffentlichen Verschlüsselungsexponenten \( e \) ab, da für sie gelten muss, dass sie teilerfremd zu \( \phi(N) \) sein müssen. In Anwendungen der realen Welt wird hier häufig \( 2^{16} + 1 = 65537 \) gewählt, im Rahmen der Wahlfreiheit, die dem Benutzer in diesem Plug-In eingeräumt wird ist aber jeder Wert möglich für den obige Bedingung gilt. Die ersten 1000 Werte werden direkt in das Drop-Down-Menü eingetragen, alle weiteren werden in einem extra Thread im Hintergrund berechnet. Das Ergebnis wird dem Nutzer über einen extra Button zugänglich gemacht, da das Eintragen der Werte in das Menü bei mehreren tausend Werten 30 Sekunden und mehr dauern kann. Nachdem der Nutzer sich ein \( e \) ausgewählt hat wird das dazu passende \( d \) berechnet, für das \( e \times d \equiv 1 \mod \phi(N) \) gilt.

### 2.3 Texteingabe

Bei der Texteingabe lassen sich zunächst zwei große Bereiche unterscheiden: Die Eingabe eines Klartextes und die eines Chiffretextes. Die Klartexteingabe unterteilt sich wiederum in Signatur und Verschlüsselung, während die Chiffretexteingabe zur Verifikation einer Signatur und zur Entschlüsselung verwendet wird.


#### 2.3.1 Klartexteingabe

Der Klartexteingabedialog wird bei der Verschlüsselung und Signaturerstellung verwendet. Bei beiden Verfahren wird der eingegebene Klartext so wie er ist in das entsprechende Feld des Plug-Ins übernommen (vgl. Abbildung 2.8). Bei der Verschlüsselung wird der eingegebene Text Buchstabe für Buchstabe in entsprechende Zahlwerte\(^2\) umgewandelt und in das entsprechende Feld eingetragen, während für die Signatur ein Hash des Textes benötigt wird, der anschließend dort eingetragen werden kann.

#### 2.3.2 Hashfunktion

Die Hashfunktion, die standardmäßig zum Einsatz kommt ist sehr einfach, aber für die Ziele des Plug-Ins vollkommen ausreichend. Zunächst werden alle eingegebenen Zeichen in ihre Zahläquiva-

\(^2\) Unicode Codepoint, vgl. [http://unicode.org/glossary/#C](http://unicode.org/glossary/#C)
Abbildung 2.6: Ablaufplan der RSA-Privatekeyerzeugung
Abbildung 2.7: Ablauf einer Runde SHA-1


Alternativ bieten die Eingabeseiten eine Checkbox an, mit der man den verwendeten Hash-Algorithmus auf SHA-1\(^3\) umstellen kann. Dieser Algorithmus arbeitet auf Blöcken von jeweils 512 Bit. Alle Blöcke werden nacheinander gehashet, wobei das Endergebnis des Vorgängerblocks die Initialisierung des nachfolgenden ist. Abbildung 2.7 zeigt eine der 80 Runden, die für jeden der Blöcke durchlaufen werden [16]. Die Werte A bis F sind Standardwerte oder das Ergebnis des letzten Blocks, die eigentliche Nachricht wird in Form der \( W_t, t \in [0, 79] \) einbezogen, die durch Kombination des zerlegten Blocks entstehen.

Da Werte von 512 Bit oder 64 Byte für diese Anwendung mit ihrem begrenzten Wertebereich jedoch viel zu groß sind, wird letztlich noch einmal der Modulo-Operator auf das Ergebnis angewendet, so dass eine Zahl zwischen 0 und \( N \) entsteht, die den Hash repräsentiert.

2.3.3 Chiffretexteingabe

Die Chiffretextdialoge haben wie die Klartextdialoge ein Feld, in das man die berechneten Zahlwerte eintragen kann. Als Eingabe werden hierbei Zahlen zur Basis 16 erlaubt, das heißt, dass sie aus Ziffern sowie den Buchstaben ’a’ bis ’f’ bestehen können. Das Leerzeichen dient als Separator der einzelnen Zahlen. Jede eingegebene Zahl wird darauf geprüft, ob sie größer als \( N \) ist. Ist dies

der Fall, wird eine Fehlermeldung angezeigt, da es dann nicht möglich ist, dass der Chiffretext mit diesem Schlüssel erzeugt wurde. Ein Fortsetzen ist erst nach Korrektur dieses Fehlers möglich.

Der Dialog zur Eingabe einer erhaltenen Signatur weist eine weitere Besonderheit auf. Hier ist es nicht nur möglich die Signatur einzugeben, sondern auch den zu dieser Nachricht gehörigen Klartext. Dieser wird schlussendlich dazu verwendet eine Prüfung der Signatur vorzunehmen und das Ergebnis im Ergebnisbereich, wie in Abschnitt 2.5 beschrieben, darzustellen.

2.4 Berechnungsanzeige und schnelles Exponentieren


Zuletzt wird noch das Ergebnis dieser Berechnung mit den eingesetzten Werten der Faktoren angezeigt und das Ergebnis zur Basis 10 sowie zur Basis 16 ausgegeben. Das Ergebnis zur Basis 16 wird auch direkt an das bisher berechnete Gesamtergebnis angehängt, so dass sich dort Schritt für Schritt das Endergebnis der Berechnung aufbaut.

Möchte man sich nicht alle Schritte anzeigen lassen, so kann man durch einen Klick auf den mit „Berechnen“ beschrifteten Button die schrittweise Berechnung überspringen und sich direkt das Endergebnis anzeigen lassen.

2.5 Ergebnisbereich


2.6 Optionen

Zweitens gibt es hier die Möglichkeit, zusätzliche Infodialoge ein- oder auszuschalten, die bei einem Klick auf die drei Hauptbuttons angezeigt werden.

Zuletzt ist hier der Reset-Button angesiedelt, der einen Abbruch des Algorithmus zu einem beliebigen Zeitpunkt ermöglicht und alle bereits eingegebenen Werte löscht.

2.7 Vergleich mit CrypTool

Auch das ursprüngliche CrypTool hat eine RSA Visualisierung. Diese bietet einen ähnlichen Funktionsumfang wie die neu entwickelte Lösung, weicht jedoch in einigen Punkten von dieser ab. Diese Punkte sollen im Folgenden genauer betrachtet und begründet werden.

Primfaktorwahl Das ursprüngliche Tool bietet keine Möglichkeit p oder q nicht prim zu wählen, es verweigert die Berechnung von N und \( \varphi(N) \) mit einer Fehlermeldung. Dadurch ist ein Fortsetzen des Algorithmus zur Schlüsselerzeugung nicht möglich. Dieses Verhalten entspricht zwar den Vorgaben des Algorithmus zur Schlüsselerzeugung, setzt aber Grenzen, da es nicht möglich ist auszuprobieren, was passiert, wenn p und q tatsächlich nicht prim gewählt werden.

Verschlüsselungsexponent CrypTool bietet hier die Eingabe beliebiger Zahlen an, erlaubt jedoch ein Fortsetzen nur, wenn die Bedingung \( \gcd(e, \varphi(N)) = 1 \) gilt. Hier ist das neue Plug-In
zwar nicht flexibler, aber anwenderfreundlicher, da es die Wahlmöglichkeiten für e vorgibt und so die ansonsten zeitaufwändige Suche nach gültigen Werten für e vereinfacht.

Gleichzeitig erlaubt das Original einen Wert für e einzugeben, der größer als $\varphi(N)$ ist und somit den Bedingungen des Algorithmus nicht entspricht, was das neue Design verhindert.

Visualisierung der Berechnung Die Berechnung mit Hilfe der schnellen Exponentierung, die einen prominenten Platz in der JCrypTool Visualisierung einnimmt, ist in Cryptool nicht vorhanden. Hier werden lediglich die Umwandlung in Zahlen und das Ergebnis präsentiert, während die Berechnung an sich außen vor bleibt.

Keystore Der Keystore ist eine Funktion, die prinzipiell in beiden Tools zur Verfügung steht. In CrypTool ist seine Verwendbarkeit jedoch beschränkt und kann nicht für die RSA-Demo verwendet werden. Im Gegensatz hierzu wird der Keystore von JCrypTool von dem neuen Plug-In in vollem Umfang genutzt und so können sowohl neu erzeugte, kleine Schlüssel, als auch mit Hilfe von FlexiProvider erzeugte Schlüssel für die Visualisierung genutzt werden.


Alphabet und Zahlensystem Die Optionen das Alphabet und Basissystem festzulegen bietet aktuell nur CrypTool, es ist jedoch mit geringem Aufwand möglich, zumindest die freie Alphabetswahl in das JCrypTool Plug-In zu übernehmen, was unter Umständen in einer weiteren Iteration geschehen wird. Auch die Einstellung der Blocklänge würde keinen übermäßigen Aufwand erfordern.

Die Variante des „Dialogs der Schwestern“[3] jedoch würde eine elementare Umgestaltung des Plug-Ins bedeuten und dabei nur einen marginalen Verständnisgewinn bringen, weshalb darauf verzichtet wird.
3 Das ElGamal Plug-In


Auch dieses Plug-In bietet dem Benutzer die Möglichkeit die meisten Schritte der Schlüsselerzeugung und -berechnung selbst zu bestimmen. Auch hier hat der Nutzer die Möglichkeit, einige Beschränkungen des Algorithmus zu umgehen und auszuprobieren, was passiert, wenn die Vorgaben nicht eingehalten werden.

3.1 Oberfläche

Nach dem Laden des Plug-Ins sieht der Benutzer die in Abbildung 3.1 dargestellte Oberfläche. Der Grundaufbau ist hier zu der Oberfläche der anderen Plug-Ins identisch, aufgrund der Anforderungen des Algorithmus existieren hier jedoch vier Buttons, die die Hauptschritte des Programmablaufs abbilden. Während der Nutzer den Algorithmus durchläuft sind immer nur die aktuelle und nächste Schaltfläche aktiv, so dass der Nutzer eine Fehleingabe korrigieren und fortschreiten kann. Wählt der Nutzer einen Punkt aus den er bereits abgeschlossen hat, dies ist anhand der grünen Umrahmung der Schaltfläche festzustellen, so werden die letzten Parameter wieder in die Eingabemaske übernommen, um so eine Modifikation ohne komplette Neueingabe zu ermöglichen. Gleichzeitig ist zu jedem Zeitpunkt ein Reset des Programms in den Ausgangszustand möglich.

3.2 Ablauf

Wie Abbildung 3.2 zeigt, verläuft die Nutzung des Plug-Ins in mehreren möglichen Pfaden, die jeweils eine der Aktionen verschlüsseln, entschlüsseln, signieren oder verifizieren darstellen. Allen gemeinsam ist, dass es jeweils möglich ist einen neuen Schlüssel einzugeben bzw. zu erzeugen oder einen bereits generierten Schlüssel aus dem JCT-Keystore zu verwenden.

Die wichtigsten Schritte hiervon sind die Schlüsselerzeugung von privaten und öffentlichen Schlüsseln, die in Abschnitt 3.2.1 erläutert wird, sowie die Eingabe von Klar- und Chiffretexten, wie sie die Abschnitte ab 3.3 aufzeigen. Im letzten Teil ab 3.6 wird schließlich gezeigt, was der Nutzer während der eigentlichen Berechnungen zu sehen bekommt, und welche weiteren Möglichkeiten für ihn existieren.

3.2.1 Ablauf der Schlüsselerzeugung

Die Schlüsselerzeugung ist daran gebunden, dass der Nutzer zunächst eine Aktion gewählt hat und damit auch bestimmt, ob er einen öffentlichen oder privaten Schlüssel erzeugen möchte. Für die Aktionen Verschlüsseln oder Verifizieren ist ein öffentlicher, für Entschlüsselung und Signatur ein
Abbildung 3.1: Startaussehen des ElGamal-Plug-Ins
Abbildung 3.2: Ablaufplan des gesamten ElGamal-Plug-Ins
privater Schlüssel notwendig. Durch einen Klick auf den mit „Schlüsselwahl“ bezeichneten Button wird der Wizard gestartet, der zur Erzeugung oder Auswahl eines passenden Schlüssels führt.

3.2.1.1 Verwendung vorhandener Schlüssel


Sollte der Nutzer sich dazu entschließen stattdessen lieber einen neuen Schlüssel zu erzeugen, so werden die entsprechenden Eingabemasken geladen, die aufgrund der Unterschiedlichkeit ihrer Eingaben und Verfahren in den folgenden Abschnitten einzeln erklärt werden.

3.2.1.2 Ablauf der Erzeugung eines öffentlichen Schlüssels

Nachdem der Nutzer sich dafür entschieden hat den öffentlichen Schlüssel manuell einzugeben, sieht er das in Abbildung 3.3 gezeigte Fenster. Dieses gibt ihm wie in Abbildung 3.4 zu sehen, auf einen gradlinigen Weg die Möglichkeit, die drei Teile des öffentlichen Schlüssels selbst einzugeben.

P wird hierbei als Primzahl gefordert, die größer als 256 ist. Damit wird sichergestellt, dass der Algorithmus auch funktionieren kann und die Buchstabenabbildung eindeutig ist. Als nächstes wird die Eingabe von g gefordert, das als Generator der zyklischen Gruppe teilerfremd zu p sein muss. Zuletzt bleibt noch die Eingabe des öffentlichen Exponenten A übrig, für den lediglich die Bedingung gilt, dass er kleiner als p sein muss.

Naturally gibt es auch die Möglichkeit diesen Schlüssel zu speichern. Dazu wird durch Auswahl einer Checkbox die Eingabe des zugehörigen Namens oder Alias auf der folgenden Seite benötigt.

3.2.1.3 Ablauf der Erzeugung eines privaten Schlüssels

Die Erzeugungsmaske für einen neuen privaten Schlüssel ist in Abbildung 3.5 zu sehen, während ein Ablaufplan, der die verschiedenen Möglichkeiten darstellt in Abbildung 3.5 zu finden ist. Grundsätzlich müssen die Werte p, G und a angegeben werden. Statt sich für eine Primzahl aus der Liste der Vorgaben zwischen 256 und 20000 zu entscheiden besteht auch die Möglichkeit, die auch in der realen Welt Anwendung findet: Man sucht sich eine Primzahl q aus und überprüft anschließend ob $2q + 1$ ebenfalls eine Primzahl ist. Gilt dieser Zusammenhang, so ist die fundene Zahl ein valides p.

Als nächstes wird die Eingabe von einem Generator g für die Gruppe benötigt. Dazu wird eine Liste der möglichen Generatoren zur Auswahl gestellt, aus denen eine Auswahl möglich ist. Zuletzt wird noch die Eingabe des „privaten Schlüssels“ a gefordert, für den gilt, dass $2 < a < p - 2$
Abbildung 3.3: Screenshot der ElGamal-Publickeyerzeugung

Abbildung 3.4: Ablaufplan der ElGamal-Publickeyerzeugung
Abbildung 3.5: Screenshot der ElGamal-Privatekeyerzeugung
Abbildung 3.6: Ablaufplan der ElGamal-Privatekeyerzeugung

gelten muss. Sobald diese Eingabe getätigt wurde, wird der zugehörige „öffentliche Schlüssel“ A angezeigt, der sich als $A = g^e \mod p$ berechnet.

Auch hier kann man durch Setzen aktivieren einer Checkbox die Speicherseite aufrufen, die die Eingabe des Besitzernamens und eines Passworts fordert und den generierten Schlüssel in den Keystore speichert.

3.3 Texteingabe

Nachdem man auf diese Art seinen Schlüssel gewählt hat geht es zur Texteingabe über. Diese gliedert sich im Wesentlichen in Klartext- und Chiffretexteingabe, auf die sich die vier möglichen Operationen verteilen.
3.3.1 Klartexteingabe

Die Klartexteingabe wird in den Fällen der Verschlüsselung und Signatur benötigt. Das Fenster sieht hierbei in beiden Fällen annähernd identisch aus, lediglich ein Button und einige Erklärungen für die Hashfunktion sind im Signaturfall zusätzlich zu finden. Das Fenster wird in Abbildung 3.7 gezeigt, während die Funktion des Buttons in Abschnitt 2.3.2 erläutert wird.

Das Eingabefenster ist recht einfach gestaltet und besteht im Wesentlichen aus einem großen Textfeld, das die Eingabe von Buchstaben und Zahlen erlaubt. Hier kann der gewünschte Klartext eingegeben oder aus der Zwischenablage eingefügt werden.

Der Klartext wird für die Verschlüsselung Buchstabe für Buchstabe in Zahlenwerte übersetzt und diese leerzeichensepariert in das entsprechende Feld der Hauptoberfläche eingetragen. Bei der Signatur hingegen wird zunächst eine Hashfunktion auf den Text angewendet, wie sie im Kapitel über RSA im Abschnitt 2.3.2 beschrieben ist. Das Ergebnis dieses Vorgangs wird ebenfalls im Textbereich der Hauptansicht angezeigt.

3.3.2 Chiffretexteingabe

Die Maske zur Chiffretexteingabe kommt bei den Varianten der Entschlüsselung und Verifika- tion zum Einsatz. Sie unterscheidet sich für die Entschlüsselung im Aussehen kaum von der Eingabemaske für die Verschlüsselung, wobei die Eingabemöglichkeiten jedoch auf leerzeichenseparierte Werte zur Basis 16 (dargestellt durch 0-9 und a-f) beschränkt sind. Diese Werte werden jeweils auf ihre Kompatibilität mit dem Modulus geprüft und nach Abschluss der Eingabe in das entsprechende Feld der Hauptansicht übertragen.

Bei der Verifikation ist die Maske zweigeteilt und bietet Eingabemöglichkeiten für Klartext und Signatur. Die Signatur besteht hierbei aus zwei Teilen, die in Klammern gesetzt und durch ein Komma getrennt sind, also die Form „( rr, ss )“ haben. Diese Eingabe erübrigt die explizite Auswahl des Parameters, der in den anderen Arbeitsmodi benötigt wird (vgl. Abschnitt 3.4). Wird zusätzlich ein Klartext eingegeben, so wird nach Ablauf der Berechnungsschritte überprüft, ob die eingegene Signatur eine gültige Signatur des Klartextes ist und eine entsprechende Meldung in der Hauptansicht angezeigt (vgl. Abschnitt 3.6).
3.4 Parameterwahl


Die Wahl dieses Parameters ist abhängig von der Aktion an unterschiedliche Bedingungen geknüpft. Bei der Ver- und Entschlüsselung ist es lediglich notwendig, dass für den mit b bezeichneten Parameter der Zusammenhang \(2 < b < p - 2\) gilt, während bei der Signatur strengere Regeln gelten. Der in diesem Fall als k bezeichnete Parameter muss sich dann im Bereich von \(0 < k < p - 1\) befinden jedoch gleichzeitig der Anforderung \(\gcd(k, p - 1) = 1\) erfüllen, also teilerfremd zu p-1 sein. Zuletzt ist die Eingabe bei der Signaturprüfung überhaupt nicht notwendig (und daher auch nicht möglich) da der Wert, den der Parameter beeinflusst als Teil der Signatur übergeben wird.

Dementsprechend ist die Eingabe dieses Parameters auch unterschiedlich gestaltet. In den einfachen Fällen der Ver- bzw. Entschlüsselung ist zusätzlich zu einem die Bedingungen nennenden Text nur ein Textfeld vorgesehen, in das der Benutzer seinen Parameter eingeben kann. Fällt dieser aus dem gültigen Bereich, so wird eine Fehlermeldung angezeigt und ein Abschluss des Wizards ist nicht möglich. Im Fall der Signatur wird der Parameter über ein vorberechnetes Auswahlfeld gewählt, das alle Werte enthält für die die oben genannten Bedingungen zutreffen. Eine Montage der beiden Möglichkeiten, die das Programm zur Parameterwahl anbietet, ist in Abbildung 3.8 zu sehen. Die Trennlinie stellt dabei die Grenze zwischen den verschiedenen Varianten, die in dieser Grafik kombiniert wurden, dar. Wie bereits gesagt, wird dieser Schritt des Programmlablaufs für die Verifikation nicht benötigt, weshalb der Button inaktiv bleibt und der Eingabewizard nicht gestartet wird.

Eine Anzeige des Parameters in der Hauptansicht findet zum jetzigen Zeitpunkt nicht statt, es ist jedoch möglich, dass dies in einer späteren Iteration des Plug-Ins hinzugefügt wird.
3.5 Berechnungsanzeige

Die Berechnungsanzeige zeigt in einfacher Textform, welche Berechnungen für einen Schritt des Kryptographieverfahrens durchgeführt werden müssen und was die Ergebnisse dieses Berechnungs- schritts sind. Das bedeutet insbesondere, dass der Parameter und die eventuellen Zwischenergebnisse wie \( B = g^b \) bei der Verschlüsselung oder \( r \) und \( s \) bei der Signatur explizit angezeigt werden. Zusätzlich werden auch die jeweiligen Berechnungsvorschriften, d.h. Formeln und Formeln mit eingesetzten Werten dargestellt, um eine Nachvollziehbarkeit der Vorgänge zu gewährleisten. Das Ergebnis des jeweiligen Schritts wird in einem eigenen kleinen Bereich angezeigt, der das Ergebnis des jeweiligen Schrittes anzeigt.

Oberhalb der eigentlichen Berechnungen befinden sich zwei Buttons: „Start“ und „Schritt“, die dazu verwendet werden, die o.g. Ergebnisse Schritt für Schritt durchzugehen und so den Aufbau des Ergebnisses mitzuverfolgen. Der Startbutton startet die schrittweise Berechnung des Ergebnisses und schaltet den Schrittbutton frei, der das Endergebnis stückweise aufbaut und jeweils den passenden Berechnungsschritt anzeigt.

3.6 Ergebnisbereich

Der Ergebnisbereich besteht im Wesentlichen aus drei Teilbereichen: Einem großen Textfeld für das Ergebnis, das bei großen Werten auch vertikal scrollbar ist, einem Feld für die Anzeige des Ergebnisses der Signaturprüfung und einem Button um das Ergebnis für die weitere Verwendung in die Zwischenablage zu kopieren.

Das Textfeld wird entweder sukzessive durch die Betätigung des Schrittbuttons aus der Berechnungsanzeige befüllt oder auf einmal mit dem kompletten Ergebnis der Berechnung durch Betätigung des „Berechnen“ Buttons aus dem Hauptbuttonbereich. Sollte das Ergebnis länger sein als eine Zeile, so wird umgebrochen, sollte es so lang sein, dass es nicht mehr in zwei Zeilen passt, so aktiviert sich die Scrollleiste, so dass das gesamte Ergebnis zugänglich wird.


Zuletzt befindet sich in diesem Bereich ein Button, der ein direktes Kopieren des Wertes in die Zwischenablage ermöglicht, um auf diese Weise eine einfache Weiterverwendung in einem eventuellen erneuten Durchlauf des Programms oder einer Speicherung für zukünftige Verarbeitung zu ermöglichen.

Das Endaussehen ist am Beispiel der Signatur in Abbildung 3.9 dargestellt. Anhand dieses Bildes lässt sich gut nachvollziehen, welche einzelnen Schritte bei dem Verfahren ablaufen und wie letztlich die Nachricht bzw. Signatur zustande kommt.

3.7 Optionen

Abbildung 3.9: Endaussehen des ElGamal-Plug-Ins

laden und benötigt zwangsläufig die Speicherung, damit das Schlüsselpaar oder die öffentlichen Schlüssel auch in den Auswahlwizards zur Verfügung stehen.
Zweitens gibt es hier die Möglichkeit, zusätzliche Infodialoge ein- oder auszuschalten, die bei einem Klick auf die drei Hauptbuttons angezeigt werden.
Zuletzt ist hier der Reset-Button angesiedelt, der einen Abbruch des Algorithmus zu einem beliebigen Zeitpunkt ermöglicht und alle bereits eingegebenen Werte löscht.

3.8 Vergleich mit CrypTool

4 Das DSA Plug-In

Der Digital Signature Algorithm DSA ist im Gegensatz zu den anderen, bereits beschriebenen Verfahren kein komplettes asymmetrisches Kryptosystem, das für Signatur/Verifikation und Verschlüsselung/Entschlüsselung verwendbar ist, sondern ein Verfahren, das lediglich eine Signaturfunktion zur Verfügung stellt. Dies ist jedoch eine Funktion, die in vielen Bereichen vollkommen ausreicht, wenn zum Beispiel die Vertraulichkeit von Daten auf andere Weise sichergestellt wird.

Als Beispiel für eine solche Situation könnte man ein Unternehmen heranziehen, das elektronische Akten führt, die auf einem verschlüsselten RAID gespeichert sind. Um nachvollziehbar zu machen, welcher Mitarbeiter welches „Blatt“ zur Akte hinzugefügt hat, bzw. wer welche Änderung durchführte, benötigt man jedoch ein kryptographisches Verfahren, das die Schutzziele Authentizität, Integrität und Nichtabstreitbarkeit sicherstellt. Gleichzeitig möchte das Unternehmen aber unter Umständen die Möglichkeit der Verschlüsselung komplett außen vor lassen. Im genannten Fall wäre dies wünschenswert, um zu verhindern, dass ein Mitarbeiter Teile der Akte verschlüsselt, wodurch die restlichen Mitarbeiter nicht mehr in der Lage wären diese Teile zu sehen oder weiter zu verarbeiten.

DSA wurde von der NSA entwickelt, 1991 patentiert und vom NIST\(^1\) als zu diesem Zeitpunkt einziger Algorithmus im DSS\(^2\) [6] empfohlen. Später traten noch die RSA Signatur und ECDSA, die Übertragung von DSA auf elliptische Kurven, hinzu [7].

4.1 Oberfläche


Auch hier liegt die bekannten Vierteilung vor, die die Oberfläche in Bereiche für Aktionsauswahl, Durchführung des Algorithmus, Informationsanzeige und Optionsbereich unterteilt. Die Buttonleiste hat vier Buttons für die verschiedenen Abschnitte des Algorithmus, wobei jedoch bei der Verifikation nur drei zum Einsatz kommen (siehe hierzu auch Abschnitt 4.4).

4.2 Ablauf

Der Ablauf des Algorithmus wie in Abbildung 4.2 scheint weniger kompliziert als bei den anderen Algorithmen, was jedoch lediglich daran liegt, dass aufgrund der fehlenden Ver- und Entschlüsselungsfunktion einige Abzweigungen im Ablaufgraph fehlen. Letztlich muss der Nutzer einen Schlüssel erzeugen oder wählen, einen Klartext oder Chiffretext (bzw. eine Signatur, optional mit da-

---

\(^1\) National Institute of Standards and Technology

\(^2\) Digital Signature Standard
Abbildung 4.1: Startaussehen des DSA-Plug-Ins
zugehörigem Klartext) eingeben und auf „Berechnen“ klicken. Danach erhält er das Ergebnis seiner Aktion, das entweder eine Signatur zum eingegebenen Text oder das Ergebnis der Verifikation einer solchen ist.

4.2.1 Ablauf der Schlüsselerzeugung


4.2.1.1 Verwendung vorhandener Schlüssel

Alternativ zur Erzeugung eines neuen Schlüssels steht dem Nutzer im Ablauf des Plug-Ins auch die Wahlmöglichkeit offen, einen bereits im Keystore vorhandenen Schlüssel zu nutzen. Sollte er diese Option wählen, so bekommt er ein Drop-Down-Menü angezeigt, aus dem er einen vorhandenen Schlüssel wählen kann, der wahlweise mit diesem Plug-In oder über die FlexiProvider Perspektive erstellt wurde.

Bei der Verwendung eines sicheren, über die FlexiProvider Perspektive erstellten Schlüssels ist jedoch zu beachten, dass die Übersichtlichkeit und Klarheit der Berechnungsanzeigen stark unter der Zahlenlänge leidet, so dass die Benutzung der kurzen Schlüssel, die dieses Plug-In generiert, empfohlen wird.

4.2.1.2 Ablauf der Erzeugung eines öffentlichen Schlüssels

Entscheidet sich der Nutzer einen neuen öffentlichen Schlüssel erzeugen zu wollen, so wird ihm die in Abbildung 4.3 gezeigte Eingabemaske präsentiert. In dieser hat er die Möglichkeit die Werte für p, q, g und y einzugeben. Diese werden dann anhand einiger Kriterien auf ihre Validität überprüft. So zum Beispiel, dass p eine Primzahl ist, $\gcd(q, p - 1) \neq 1$ gilt und q, g und y jeweils positiv und kleiner als p sind. Sollte eine oder mehrere dieser Bedingungen nicht erfüllt sein, so wird eine Fehlermeldung angezeigt und ein Abschluss des Wizards verhindert.

Wenn alles korrekt ist, ist es dem Nutzer möglich den Schlüssel zu speichern, wofür er aufgefordert wird einen Namen einzugeben oder ohne Speichern fortzufahren, wobei der Schlüssel nach dem einmaligen Durchlauf des Plug-Ins mit dem Reset verloren geht.

4.2.1.3 Ablauf der Erzeugung eines privaten Schlüssels

Abbildung 4.2: Ablaufplan des gesamten DSA-Plug-Ins
Abbildung 4.3: Screenshot der DSA-Publickeyerzeugung
Abbildung 4.4: Screenshot der DSA-Privatekeyerzeugung

Aufgrund der Bedingung, dass N kleiner oder gleich der Ausgabenlänge der Hashfunktion sein muss, besteht hier eine generelle Beschränkung auf Werte kleiner oder gleich 160, und sogar kleiner oder gleich 8, sofern die vereinfachte Hashfunktion verwendet werden soll. Ist das gewählte N größer als 8, so wird eine Warnung eingeblendet, die auf diesen Umstand hinweist. Für L gelten keine besonderen Einschränkungen, außer dass \( L > N \) gelten muss, wenngleich auch hier große Werte die Berechnungsanzeige unnötig verkomplizieren, so dass auch hier kleine Werte empfohlen werden.

Sobald der Nutzer diese Auswahl getroffen hat bekommt er eine Liste von möglichen Primzahlen q mit der eingegebenen Bitlänge angezeigt, aus denen er eine Zahl auswählt. Ist dies geschehen, so werden die möglichen Primzahlen für p berechnet und angezeigt für die gilt, dass p-1 ein Vielfaches von q ist.

Nachdem auch diese Zahl gewählt ist, kann ein beliebiges h zwischen 1 und p-1 gewählt werden, aus dem sich g als \( g = h^{\frac{p-1}{q}} \mod p \) berechnet. Sollte sich hierbei g = 1 ergeben, so wird eine Fehlermeldung mit der Bitte ein neues h zu wählen angezeigt. Im Allgemeinen sollte jedoch der Großteil der möglichen Werte für h ein gültiges Ergebnis liefern.
Zuletzt wird noch ein Wert \( x \) zwischen 1 und \( q \) gewählt, der den eigentlichen privaten Schlüssel bildet, während die anderen Komponenten prinzipiell auch für verschiedene Personen verwendet werden können. Sobald \( x \) gewählt wurde, wird der letzte Teil des öffentlichen Schlüssels \( y = g^x \mod p \) berechnet und alle Werte werden mit Abschluss des Wizards in die Hauptansicht eingetragen.

### 4.3 Texteingabe

Die Texteingabe läuft prinzipiell so ab wie bei den Signaturfunktionen der anderen Plug-Ins auch: Es wird für die Signatur ein Klartext und für die Verifikation eine Signatur der Form \((r, s)\), sowie optional der zugehörige Klartext eingegeben. In beiden Fällen ist abhängig von der Größe des gewählten \( N \) die Möglichkeit gegeben, die Hashfunktion zu bestimmen\(^3\), indem eine Checkbox wird. Die vereinfachte Hashfunktion, die bei hinreichend kleinem \( N \) ausgewählt werden kann, ist in Abschnitt 2.3.2 ausführlich erklärt, weshalb hier auf eine weitere Erläuterung verzichtet wird.

### 4.4 Parameterwahl

Die Ähnlichkeit zwischen ElGamal und DSA spiegelt sich auch im nächsten Schritt wieder, der allerdings nur bei der Signatur einer Nachricht durchzuführen ist: Es muss ein Parameter gewählt werden, der, weil er nur einmal benutzt werden sollte, dafür sorgt, dass die Signatur derselben Nachricht nie dasselbe Ergebnis liefert. Aufgrund der Tatsache, dass dieser Schritt für die Verifikation nicht notwendig ist, da der Parameter bereits implizit mit der Signatur übergeben wird, bleibt der Button inaktiv und wird optisch direkt auf abgeschlossen gesetzt, indem der Rahmen grün eingefärbt wird.

Problematisch gegenüber ElGamal ist hierbei jedoch, dass \( k \) aus einem deutlich kleineren Wertebereich stammt, weil \( 1 < k < q \) gelten muss. Dies führt unter Verwendung der einfachen Hashfunktion selbst im Idealfall dazu, dass es weniger als 256 mögliche \( k \) gibt.

### 4.5 Berechnungsanzeige

Die Berechnungsanzeige ist auch bei diesem Plug-In das Herzstück der Anwendung. Hier wird ähnlich wie bei ElGamal dargestellt, welche Schritte ausgeführt werden, um zur Signatur zu gelangen, bzw. was berechnet wird um zu verifizieren, dass die angegebene Signatur auch wirklich zum eingegebenen Klartext passt.

### 4.6 Ergebnisbereich

Der Ergebnisbereich zeigt das Endergebnis der Signatur oder Verifikation an. Im Falle der Signatur besteht der Inhalt aus einem Zahlenpaar der Form \((r, s)\), wobei \( r \) und \( s \) die beiden Werte sind, die in der Berechnungsanzeige als \( r = (g^k \mod p) \mod q \) bzw. \( s = k^{-1}(H(m) + x \cdot r) \mod q \) berechnet wurden. Im Falle der Signatur befindet sich in diesem Bereich nur der letztlich berechnete Wert \( v \), der mit dem \( r \) der Signatur übereinstimmen muss, damit diese gültig ist.\(^3\) nur wenn \( N \leq 8 \)
Neben diesem Feld existiert ein weiteres, das nur im Fall der Verifikation Verwendung findet. Es zeigt ein grünes „gültig“ oder rotes „ungültig“ an, je nachdem ob die Signaturprüfung erfolgreich war oder fehlgeschlagen ist.

Das letzte Element im Ergebnisbereich ist schließlich der mit „kopieren“ beschriftete Button, der es ermöglicht, das Ergebnis eines Vorgangs (in der Regel die Signatur) direkt in die Zwischenablage zu kopieren und sie z.B. nach einem Reset des Plug-Ins für die Verifikation weiter zu verwenden.

### 4.7 Optionen

Der Optionsbereich ist mit dem der anderen Plug-Ins identisch und bietet die Möglichkeiten, das Plug-In per Klick auf „Reset“ zurückzusetzen, zusätzliche Informationsdialoge ein- oder aus zu schalten und einen eigenständigen Schlüsselerzeugungswizard zu starten, mit dem neue Schlüssel erzeugt werden können, ohne den aktuellen Zustand des Plug-Ins zu verändern.

### 4.8 Vergleich mit CrypTool

5 Das Dummy Plug-In

Bei der Entwicklung der Plug-Ins für die einzelnen Verschlüsselungsverfahren stellte ich fest, dass der grundsätzliche Aufbau und einige Funktionen letztlich bei jedem der drei Plug-Ins identisch waren. Daher entschied ich mich diese gemeinsamen Komponenten und Funktionen in ein eigenes Plug-In auszulagern, das zukünftigen Entwicklern die Möglichkeit geben sollte ein Grundgerüst zu haben, auf dem sie aufbauen können.

So entstand das Dummy Plug-In, das in Abbildung 5.1 dargestellt wird. Auf diesem Screenshot sind auch kurze Erklärungen zu finden, für was welcher Bereich gedacht ist bzw. verwendet werden kann. Die Buttons sind bereits mit der Funktion eines Wizardaufrufs belegt, welcher eine Dummyseite anzeigt, um Entwicklern auch dieses Konzept einfach nahe zu bringen.

Abbildung 5.1: Aussehen des Dummy Plug-Ins

Die Verwendung dieses Prototyps für zusätzliche Plug-Ins stellt, sofern sich die Algorithmen mit dieser Aufteilung darstellen lassen, eine Kontinuität im Programm dar, die es einem Nutzer ermöglicht, einfach zwischen verschiedenen Algorithmen zu wechseln und sich nicht in jeden neu eindenken bzw. einarbeiten zu müssen.
Abbildung 5.2: Aussehen des Dummy Wizards
6 Interessante Codeschnipsel


6.1 Sub- und Superscript

Die Darstellung von Sub- und Superscripten, unter Umständen auch doppelten Hochstellungen, stellt in der aktuellen Version des SWT noch immer ein Problem dar. Es gibt keine direkte Möglichkeit einen formatierten Text so in ein Textfeld einzutragen, wie ihn zum Beispiel das \LaTeX-Kommando $110^{2^k}$ bieten würde.

Es bleiben einem prinzipiell zwei Möglichkeiten:

1. die eben verwendete Darstellung als Text zu verwenden, was den meisten Naturwissenschaftlern geläufig sein dürfte oder

2. den komplizierteren Weg zu gehen und die Hochstellung manuell zu erzeugen, was auch Laien das Verständnis ermöglicht.

Für das vorliegende Plug-In wurde der zweite Weg gewählt, da man so eine Darstellung erhält, die weniger Vorwissen der Benutzer voraussetzt.

Um ein Sub- oder Superscript darzustellen, benötigt man einige Grundangaben, die in Listing 6.1 dargestellt sind:


2. Daraus erstellt, je ein Font- und FontData-Objekt, das Name, Größe und Stil der verwendeten Schrift beschreibt.


4. Und zuletzt TextStyle-Objekte, die die Möglichkeit bieten, über das enthaltene rise-Feld die Grundlinie der Schrift zu verschieben.

```java
1 // get the graphics context
final GC gc = new GC(fastExpTable);
2 // get the standard font we’re using everywhere
final Font normalFont = getDisplay().getSystemFont();
3 // get the associated fontData
final FontData normalData = normalFont.getFontData()[0];
```
// set the new font height to 12
normalData.setHeight(12);
// create small and very small data from the normal data and create matching fonts with each 2pt less height
final FontData smallData = new FontData(normalData.getName(), normalData.getHeight() - 2, normalData.getStyle());
final Font smallFont = new Font(getDisplay(), smallData);
final FontData verySmallData = new FontData(smallData.getName(), smallData.getHeight() - 2, smallData.getStyle());
final Font verySmallFont = new Font(getDisplay(), verySmallData);
// something to calculate the actual place of the superscripts
final int baseline = gc.getFontMetrics().getAscent() + gc.getFontMetrics().getLeading();
// small and very small text styles for the super— and subscripts
TextStyle superScript = new TextStyle(smallFont, null, null);
TextStyle superSuperScript = new TextStyle(verySmallFont, null, null);
TextStyle subscript = new TextStyle(verySmallFont, null, null);
// rises, values found by experiment
superScript.rise = baseline / 2 - 1;
superSuperScript.rise = baseline - 2;
subscript.rise = -baseline / 2 + 2;

Listing 6.1: Grundangaben für Superscripts

Mit diesem Rüstzeug ausgestattet ist es nun möglich, an (nahezu) beliebigen Stellen, Hoch- und Tiefstellungen zu erzeugen, auch wenn dies selbst mit etwas Aufwand verbunden ist, wie das folgende Listing 6.2 am Beispiel einer Zeile zur Erklärung des schnellen Exponentierens zeigt.


Nachdem nun die Vorbereitungen abgeschlossen sind, wird der eigentliche Inhalt in Form eines TextLayout-Objekts erzeugt. Diesem Objekt wird als Erstes der darzustellende Text mitgeteilt, und anschließend in welchem Bereich welcher Style angewendet werden soll. Analog zu den gezeigten Hochstellungen sind auch Tiefstellungen möglich, indem das entsprechende subscript-Objekt verwendet wird oder die rise-Felder angepasst werden.

// drawing area
Composite styledFastExpMulText = new Composite(g, SWT.NONE);
// create the TextLayout
TextLayout styl0r = new TextLayout(getDisplay());
// paint listener
styledFastExpMulText.addListener(SWT.Paint, new Listener() {
  public void handleEvent(final Event event) {
    styl0r.draw(event.gc, 0, 0);
  }
});
// set the text which will be displayed as 72^{97} = 72^{20} \cdot 72^{25} \cdot 72^{26}
styl0r.setText("7297\underline{\underline{7220}} \cdot \underline{7225} \cdot \underline{7226}");
6.2 Threading

Threading ist eine einfache Möglichkeit komplizierte oder langwierige Aufgaben, die keine Interaktion mit dem Nutzer benötigen, in einen eigenen Programmmstrang auszulagern. Gerade für die heutigen und zukünftigen Multicore- und Multiprozessorsysteme ist Threading ein elementarer Baustein, um die Kapazitäten des Systems ausnutzen zu können. Versucht man jedoch in einer RCP-Anwendung einen eigenen Thread für Berechnungen zu verwenden und die Ergebnisse anschließend in ein UI-Element einzutragen, so wird man schnell auf eine IllegalThreadAccessException stoßen, sofern man nicht wie im Folgenden beschrieben vorgeht.


```java
new Thread() {
    public void run() {
        Set<BigInteger> tempEList = new TreeSet<BigInteger>();
        BigInteger ih;
        for (int i = 2; i < phin.intValue(); i++) {
            ih = new BigInteger("" + i);
            if (phin.gcd(ih).equals(ONE)) {
                if (tempEList.size() == KeyRunnable.TRIGGERLENGTH) {
                    fillToE(tempEList, true);
                }
                tempEList.add(ih);
            }
        }
        fillToE(tempEList, false);
    }
}
```
private void fillToE(Set<BigInteger> list, boolean intermediate) {
    List<String> newList = new LinkedList<String>();
    for (BigInteger integer : list) {
        newList.add(integer.toString());
    }
    Display.getDefault().asyncExec(
        new KeyRunnable(newList.toArray(new String[newList.size()]),
            intermediate));
}.start();

Listing 6.3: Threading in SWT

Dieser erste Thread wird gestartet, sobald die Bedingungen für die Berechnung der e-Werte gegeben sind. Die Ergebnisse werden dann zu zwei Zeitpunkten mit einem in Listing 6.4 gezeigten KeyRunnable an die oben genannte Methode asyncExec übergeben, um die Eintragung in die UI zu ermöglichen.

Der erste Zeitpunkt für die Erzeugung einer solchen KeyRunnable ist, sobald eine in der Variablen TRIGGERLENGTH festgelegte Anzahl Werte für e gefunden wurde. Dieses Zwischenergebnis wird dann direkt in das Auswahlfeld eingetragen. Der zweite Aufruf findet statt, sobald die Liste fertig berechnet ist, wobei die Liste dann in den data-Bereich des Eintragungbuttons gespeichert und erst bei Klick auf diesen in die Auswahlliste übernommen wird.

```java
final class KeyRunnable implements Runnable {
    private static final int TRIGGERLENGTH = 1000;
    private final String[] newList;
    private final boolean intermediate;
    
    private KeyRunnable(String[] list, boolean intermediate) {
        this.newList = list;
        this.intermediate = intermediate;
    }

    public void run() {
        if (newList.length <= TRIGGERLENGTH) {
            elist.setItems(newList);
            elistUpdate.setData(null);
            elistUpdate.setEnabled(false);
            elistUpdate.setVisible(intermediate);
        } else {
            elistUpdate.setData(newList);
            elistUpdate.setEnabled(true);
        }
    }
}
```

Listing 6.4: Eintragungsrunnable
7 Rück- und Ausblick

Die vorliegende Dokumentation beschreibt, was unternommen wurde, um JCrypTool näher an die Funktionen von CrypTool heranzubringen um einen vollwertigen Nachfolger zu erhalten. Gleichzeitig wurden die bestehenden Funktionen erweitert um das Verständnis für cryptographische Algorithmen auch Personen zu ermöglichen, die mit der verwendeten Mathematik im Allgemeinen und Cryptographie im Besonderen nicht weitergehend vertraut sind.

Hierzu wurden für die Standardalgorithmen RSA, DSA und ElGamal Plug-Ins geschrieben, die die einzelnen Berechnungsschritte möglichst einfach darstellen und zeigen, wie man vom Klar- zum Chiffretext und zurück gelangt, bzw. eine Signatur erzeugt oder verifiziert. Auch das Verständnis für die Aufgabe und Bedeutung einer Hashfunktion für die Signatur kann verbessert werden, da eine Hashfunktion implementiert wurde, die auf eine einfach nachvollziehbare, wenngleich unsichere Weise arbeitet.

7.1 Ausblick


Zusätzlich sind die Algorithmen aktuell auf das Alphabet von a-z und A-Z sowie die kleinen und großen Umlaute ä, ö, ü und Ä, Ö, Ü beschränkt. Dieses vorgegebene Alphabet könnte durch ein spezifisch durch den User festlegbares ersetzt werden, was entweder die notwendige Größe verschiedener Parameter reduzieren würde oder die Möglichkeit bieten könnte beliebige Texte auch in Sprachen mit nicht-lateinischem Alphabet zu verarbeiten.
Literaturverzeichnis


