Technische Dokumentation

Running Key / JCrypTool

Martin Niederwieser, Georg Chalupar, Simon Scheuchenpflug

Semesterprojekt WS 2010/11

Sichere Informationssysteme Bachelor

FH Oberösterreich Studienbetriebs GmbH

Fakultät für Informatik, Kommunikation und Medien

Softwarepark 11, 4232 Hagenberg, Austria

http://www.fh-hagenberg.at

INHALTSVERZEICHNIS

ALLGEMEIN

JCRYPTOOL

JCrypTool ist eine aufstrebende kryptographische E-Learning-Plattform. Entwickelt wird sie auf Basis der Eclipse Rich Client Platform (RCP). Sie ist ein Open Source Projekt, die es Studenten, Lehrern, Entwicklern und anderen Kryptographie Interessierten ermöglicht die kryptographischen Mechanismen zu verstehen bzw. in modernen leicht zu verstehenden Anwendungen auszuprobieren. Ziel ist es eine neue Form von E-Learning zu schaffen. Nicht nur stupides auswendig lernen, sondern einfaches verstehen und anwenden. Es soll Anwendern ermöglichen auch selbst kryptographische Plugins zu entwickeln und diese in JCrypTool einzubinden. Momentan sind schon verschiedene Mechanismen implementiert z. B.: symmetrische- bzw. asymmetrische Verschlüsselung, kryptographische Analysen, Hashwerte oder auch Spiele.

JCrypTool ist der Nachfolger der bekannten und sehr beliebten E-Learning Plattform CrypTool. JCrypTool enthält die schon lange bewährten Bibliotheken BouncyCastle und FlexiProvider (FlexiProvider ist der Standard „crypto provider“). Andere fortgeschrittenen Bibliotheken sind auch möglich; die Benutzer können ihren gewünschten Provider wählen.

PROJEKTZIELE UND HINTERGRÜNDE

Ziel war die Entwicklung eines JCrypTool-Plugins zur Analyse und Visualisierung des Running Key Ciphers.

Ein Running Key Cipher ist das Ergebnis der Verknüpfung zweiter Klartexte.

Selbiges Problem tritt bei der Verwendung desselben Schlüssels zum Verschlüsseln verschiedener Texte auf. Durch ein einfaches XOR wird es nämlich möglich den Schlüssel herauszukürzen und man erhält wieder die Verknüpfung der Klartexte, einen Running Key Cipher.

Plaintext A	1010
Key	0110
Cipher A	1100

Plaintext B	1101
Key	0110
Cipher B	1011

Cipher A	1100
Cipher B	1011
XOR	0111

Plaintext A	1010
Plaintext B	1101
XOR	0111

Veranschaulichung eines Running Key Ciphers

EINRICHTUNG

JCrypTool verwendet Eclipse RCP SDK als Entwicklungsumgebung. Zum Entwickeln des Plugins wurde die Version 3.6.1 verwendet. Zum Einbinden des JCrypt- und des FH-Repositories musste zusätzlich das Plugin Subversive installiert werden.

ECLIPSE

Eclipse Helios wurde wie in der JCrypTool –Wiki beschrieben installiert und eingerichtet [1].

Da am Anfang des Projekts eine Umstellung auf die neue Eclipse-Version vollzogen wurde, kam es zu einigen Einrichtungsschwierigkeiten. In der Zwischenzeit ist die Wiki aber wieder aktualisiert worden und ein Einrichten mit den genannten Schritten sollte fehlerfrei möglich sein.

Probleme bereitete das, in der Wiki unter “Setting up a target platform” zu findende, “equinox”-File. Ohne das Einbinden dieses neuen Files kommt es zu Fehlermeldungen in Eclipse.

Sollte nach dem Einbinden der neuen “equinox”-Datei noch immer ein Fehler beim Starten erscheinen (Errorcode 13), hilft es im Eclipse Ordner die Datei “eclipse.ini” zu öffnen.

Durch Vergrößern der Werte “Xms” und “Xmx” teilt man Eclipse mehr Speicher zu, was in unserem Fall ein korrektes Ausführen ermöglichte.

```
-startup
plugins/org.eclipse.equinox.launcher_1.1.0.v20100507.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.1.1.R36x_v20100810
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
-vmargs
-Xms512m
-Xmx1024m
```

Lauffähiges “eclipse.ini” File

Sollte das File korrekt eingebunden und alle in der Wiki erwähnten Schritte ausgeführt worden sein, lässt sich JCrypTool mit der Datei jcryptool.product im Ordner org.jcryptool.core starten.

[1]JCrypTool Wiki für Entwickler. URL: http://sourceforge.net/apps/mediawiki/jcryptool/index.php?title=GettingStartedDeveloper (abgerufen am 25.02.2011)

SUBVERSIVE

Addons in Eclipse kann man im Menü unter Help -> „Install new Software“ suchen und installieren.

Nach der Installation des Plugins, kann man es sich mit dem folgenden Befehl einblenden: Window -> Show View -> Other … -> SVN -> SVN Repositories

Ein eigener Reiter SVN-Repositories sollte nun erschienen sein. Mit Rechtsklick -> new -> Repository Location ist es möglich den Pfad zum jeweiligen Repository (JCrypt oder FH) anzugeben.

Für das Entwickeln mit den JCrypTool Sourcen mussten zwei Repositories eingebunden werden:
https://jcryptool.svn.sourceforge.net/svnroot/jcryptool
https://jctplugins.svn.sourceforge.net/svnroot/jctplugins

Um die zum Arbeiten benötigten Sourcen herunterzuladen, müssen die Repositories ausgecheckt werden (Rechtsklick auf das Repository -> Check Out).

ALGORITHMUS

BESCHREIBUNG

Prinzipiell stand uns der zu verwendende Algorithmus offen.
Nach langer Internetrecherche stellten wir jedoch fest, dass der Viterbi Algorithmus der einzige Algorithmus mit zufriedenstellenden Ergebnissen für die Lösung eines Running Keys ist.

Alternative Häufigkeitsanalysen lieferten in etwa nur 30% korrekte Buchstaben.

Als Hauptinformationsquelle zum Verständnis des Algorithmus diente ein Paper des Royal Institutes of Technology in Schweden [1].

NGRAMMME

Ein NGramm repräsentiert eine Wahrscheinlichkeit einer bestimmten Zeichenfolge.

Ein NGramm-Modell besteht aus unzähligen NGrammen und dient als Ausgangslage für die weiteren Berechnungen des Viterbi-Algorithmus. Als Grundbedingung zum Lösen eines Running Key Ciphers muss das Wissen um die Sprache der Klartexte vorhanden sein. Aus diesem Wissen wird dann ein Sprachmodel für jede Sprache erstellt. Dabei ist es egal ob es sich um eine „herkömmliche“ gesprochene Sprache oder um eine Programmiersprache, wie zum Beispiel html, handelt. Je stärker die Typisierung einer Sprache desto aussagefähiger ist das Modell.

---

[1] Ekera, M., B. Terelius: Automatic solution in depth of one time pads, März 2008. URL: http://www.csc.kth.se/utbildning/kth/kurser/DD2449/krypto09/pdf/rapport_vernam080309.pdf (29.01.2011 16:40)

Komö|26|	uipm|2|
Komöd|26|	uipme|2|
Kon|341|	uir|36|
Konc|2|	uire|19|
Konce|2|	uired|1|

Als Erstes sieht man die entsprechende Zeichenkette und zwischen den „|“-Zeichen ihr absolutes Vorkommen in allen eingelesenen Texten bei der Erstellung.

Auszüge aus dem deutschen Sprachmodell

Im JCrypTool enthalten sind vorgefertigte Sprachmodelle für deutsche und englische Texte.

Bei der Erstellung der NGramme ist zu überlegen, bis zu welcher Zeichenlänge NGramme sinnvoll sind. Mit größeren NGrammen bzw. ihren Wahrscheinlichkeiten wird das Ergebnis zwar besser, jedoch steigt der Speicherverbrauch exponentiell an. In der Praxis haben sich 5-Grammme bewährt.

VITERBI ALGORITHMUS

Grundsätzlich ist der Viterbi Algorithmus eine Art der Häufigkeitsanalyse.
Man versucht mithilfe der Wahrscheinlichkeiten, für das Vorkommen von Zeichenketten, den Klartext zu rekonstruieren.

Die Arbeitsweise lässt sich grob in folgende Schritte einteilen:

1. Ein Buchstabe wird in alle möglichen Kombinationen der Verknüpfung zerlegt. Als Umkehrfunktion verwendet man beim klassischen XOR-Verfahren ein weiteres XOR und bei der modularen Addition eine modulare Subtraktion.
2. Es wird die Wahrscheinlichkeit für jede dieser Kombinationen berechnet. Diese Berechnungen erledigt ein komplexes Sprachmodell, welches unterschiedlich lange Zeichenfolgen berücksichtigt. Wie viele Vorgängerbuchstaben in die Berechnung mit einfließen, hängt von der eingestellten N-Gramm-Größe ab.
3. Die wahrscheinlichsten Pfade werden geordnet gespeichert. Unwahrscheinliche Pfade werden nicht weiterverfolgt und vom Algorithmus verworfen. Dieser Punkt ist speziell für die Laufzeit des Algorithmus entscheidend, da diese sonst einem BruteForce-Angriff gleichkommen würde.

Jedoch ist dies nur eine sehr grobe Darstellung.
Eine wesentlich detailliertere Ablaufbeschreibung findet man in den JCrypTool Sourcen unter:
org.jcryptool.visual.viterbi\src\org\jcryptool\visual\viterbi\algorithm

RESULTATE

Klartext 1:
Give in like a good fellow, and bring your garrison to dinner, and beds afterwards. Nobody injured, I hope?

Klartext 2:
Deeply regret advise your Titanic sunk this morning fifteenth after collision iceberg resulting serious los[s life furhter particualrs.]
(Text in [ ] wird abgeschnitten)

XOR-Verknüpfung:
03 0c 93 95 cc 90 4e d2 09 8e 99 00 d4 41 41 03 99 06 17 c5 c6 9c 09 99 1d d7 78
c9 95 0f 0a c9 81 d2 9a 1b 09 4b 59 1b 1d 1b 53 47 0c 1d 00 87 9a 81 09 00 12 06
c6 90 0c 8b 00 11 9a 0c 41 87 9a 81 d2 42 06 8b 9f cc 88 95 1d 0a 9c d7 88 11 81
11 4b d2 a9 cf 90 0a 17 0c cc 1d 87 84 12 d2 96 81 de c9 a6 55 1b cf 9c 0a cc 53

Lösung 1:
wery in like a good fellow, and bring your garrison fifteenth after collision iceberg resulting serious los

Lösung 2:
tially regret advise your Titanic sunk this morning to dinner, and beds afterwards. Nobody injured, I hope? later

Auswertung:

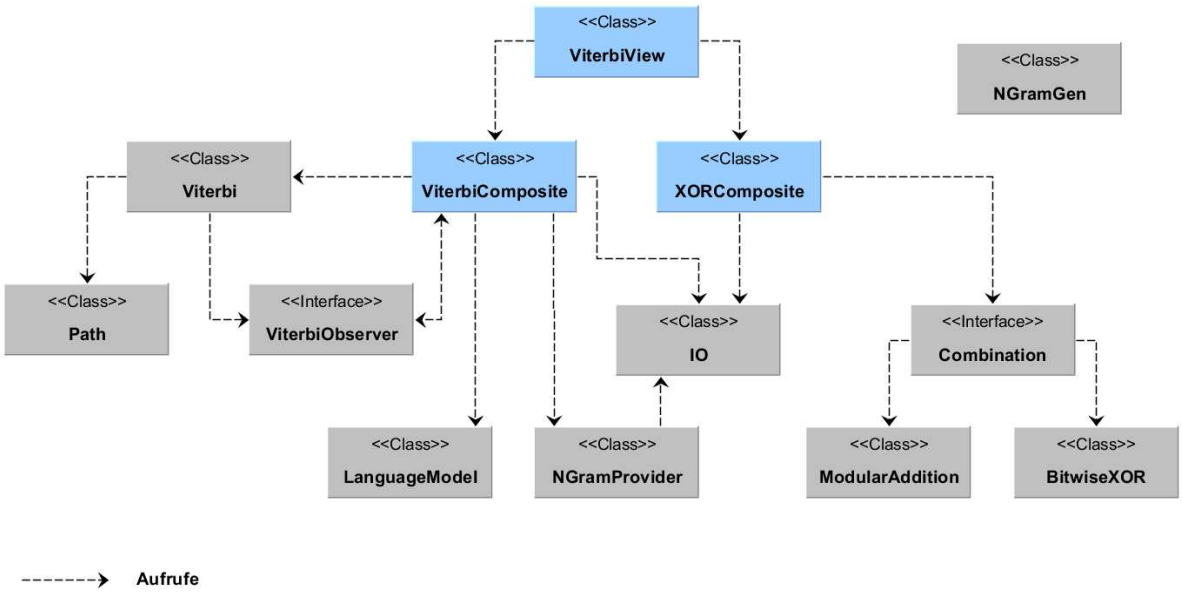
Korrekt gelöst: 97.19 % (ohne Berücksichtigung von Sprüngen zwischen den Lösungen)
Vollständig korrekt: 51.40 %
Textlänge: 107 Zeichen

Das erste rot markierte Wort der in beiden Lösungen wurde falsch rekonstruiert. Der Anfang des Algorithmus ist am Fehleranfälligsten, da hier noch nicht viele Informationen gesammelt werden konnten.

In der Mitte des Analysetextes befindet sich ein Sprung. Wenn an der gleichen Stelle in beiden Klartexten das gleiche Zeichen steht, kann der Algorithmus nicht unterscheiden, ob sich ein entschlüsselter Buchstabe im ersten oder im zweiten Text befindet. Die zwei verschiedenen Farben sollen den Sprung verdeutlichen.
Der Text bleibt aber weiterhin lesbar und die mögliche Anzahl von Sprüngen beschränkt.

STRUKTUR

Das Package org.jcryptool.visual.viterbi.algorithm besteht aus insgesamt 8 Klassen und 2 Interfaces.



PROGRAMMMIER-ELEMENTE

Je größer die verwendeten NGrammme werden, desto öfter passiert es, dass im Klartext vorkommende Zeichenketten nicht in den vorberechneten NGrammen vorkommen. Bei 7-Grammen müssten etwa alle Wörter mit 7 oder weniger Buchstaben komplett in den eingelesenen Texten zur NGramm-Berechnung enthalten sein. Dies ist natürlich bei Eigennamen und Fachbegriffen so gut wie nie der Fall.

Um seltene, aber dennoch der Charakteristik der verwendeten Sprache entsprechenden Zeichenfolgen erraten zu können gibt es mehrere Ansätze. Ein einfacher Ansatz wäre für jedes nicht eingelesene NGramm eine fixe Wahrscheinlichkeit kleiner der Wahrscheinlichkeit des seltensten NGramms vorzusehen.

Im Viterbi-Algorithmus werden Wahrscheinlichkeiten multipliziert, daher sind Wahrscheinlichkeiten von 0 besonders schlecht, da sich ein Pfad davon nicht mehr "erholen" könnte. Wenn alle Pfade auf 0 abfallen, geht überhaupt die Unterscheidbarkeit von guten und schlechten Pfaden verloren. Daher kann selbst dieser primitive Ansatz die Ergebnisse verbessern. Jedoch fließen dabei keine Informationen des Sprachmodells ein, was dazu führt, dass der Begriff "Rhododendron" die gleiche Wahrscheinlichkeit hat wie "y0T4HKzMMkZY", vorausgesetzt beide waren nicht in den NGramm-Quellen vorhanden.

Der von uns verwendete, optimierte Ansatz berücksichtigt zusätzlich die Wahrscheinlichkeit von Teilen der Zeichenfolge. Das führt dazu, dass Wörter, die der Charakteristik der Sprache entsprechen eine vergleichsweise hohe Wahrscheinlichkeit zugeordnet bekommen, auch wenn sie so nicht eingelesen wurden. Zufällige Zeichenfolgen werden weiterhin äußerst geringe Wahrscheinlichkeiten zugeordnet.

Die Wahrscheinlichkeit, dass das Zeichen x_n auf die Zeichenfolge x_m^{n-1} folgt, wird berechnet durch:

$$\hat{p}(x_n \mid x_m^{n-1}) = \begin{cases} |A|^{-1} & \text{if } c(x_m^{n-1}) = 0 \\ c(x_m^n)/c(x_m^{n-1}) & \text{otherwise} \end{cases}$$

$$P(x_n \mid x_m^{n-1}) = \sum_{j=m}^{n-1} a^{j-m} \cdot (1-a) \cdot \hat{p}(x_n \mid x_j^{n-1}) + a^{n-m} \frac{c(x_n) + \lambda}{\sum_{y \in A} c(y) + \lambda \cdot |A|}$$

Der Parameter a sagt dabei aus, wie groß der Anteil an fehlenden nGrammen ist. λ (Lamda)ist der erwartete Anteil an Zeichen, die nicht in den nGramm-Quellen vorhanden waren.

Durch den Faktor $a^{(j-m)}$ werden die kürzeren nGramme schwächer gewichtet.

GUI/JCRYPTOOL-FRAMEWORK

Aus Referenzmodell für unsere Visualisierung diente das bereits bestehende RSA Plugin, da dieses bereits die von uns benötigten Tab integriert hatte und der Code gut leserlich war.

BESCHREIBUNG

Das GUI des Plugins ist in zwei Tabs unterteilt, die untereinander Werte für Berechnungen austauschen.

Tab eins dient der Berechnung unseres Running Key Ciphers. Der User hat dabei die Möglichkeit die beiden Klartexte manuell einzugeben, oder über eigene Buttons Textdateien zu laden. Als Verknüpfungsoperationen wurden dabei das klassische XOR, sowie die modulare Addition implementiert.

Da bei der Erzeugung eines Running Key Ciphers auch nicht druckbare Zeichen enthalten sein können, werden diese in der Textdarstellung entsprechend maskiert, bzw. ist aus diesem Grund auch die Möglichkeit einer Hexadezimaldarstellung gegeben.

Mit einem Klick auf den Button „Übernehmen“ wird der Wert des Ciphertextfeldes in das entsprechende Feld im nächsten Tab geschrieben und der Tab gewechselt.

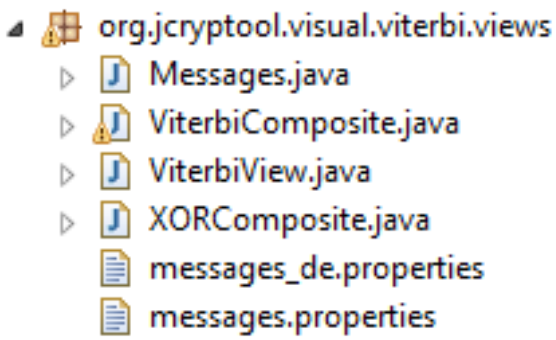
Der Tab Viterbi enthält die Darstellung des eigentlichen Algorithmus. Neben der Sprache hat der Benutzer auch die Möglichkeit NGramm-Größe und Pfadanzahl einzustellen, wobei höhere Werte zu einem besseren Ergebnis führen sollten.

Der Algorithmus ist im Plugin durch Threads von der GUI getrennt, was bei der Berechnung eine Liveausgabe der wahrscheinlichsten Lösungen ermöglichte.

Aus Komfortgründen wird dem Start-Button während der Laufzeit des Algorithmus zusätzlich eine Abbruchfunktion gegeben.

STRUKTUR

Die Elemente der GUI liegen im Package „org.jcryptool.visuals.viterbi.views“.



Messages.java
Die Klasse enthält die statischen Variablen, mit denen in den jeweiligen Sprachfiles nach den Texten gesucht wird.

messages_de.properties/messages.proberties
In diesen beiden Files werden den in Messages.java definierten statischen Variablen entsprechende Texte zugewiesen, die dann in der GUI sichtbar werden.

ViterbiView.java
Klasse zum Erstellen des Tab Menüs und der Funktion zum Wechseln der Tabs. Den beiden Tab Items wird dabei mit nur mit Messages.XYZ der Name übergeben. Der eigentliche Inhalt der Tabs wird dem ScrolledComposite als Objekt übergeben.

XORComposite.java
Enthält den die GUI-Elemente des XOR Tabs.

ViterbiComposite.java
Enthält die GUI-Elemente des Viterbi-Tabs und greift auf den Algorithmus zu.

LAYOUT UND GUI-ELEMENTE

Für die grafische Struktur der Tabs in der GUI wurden verschachtelte GridLayouts verwendet. In SWT gibt es mehrere GUI-Elemente, die als Container für andere GUI-Elemente dienen. Solche Container sind von Composite abgeleitet, Beispiele sind Group und Canvas. Um festzulegen wo die enthaltenen Elemente eines Canvas am Bildschirm angezeigt werden sollen gibt es Layouts. Das Layout ist eine Eigenschaft des Containers und wird durch setLayout() gesetzt.

Das GridLayout unterteilt die Fläche in ein Raster, das zeilenweise befüllt wird. Um ein Element in einen Container hinzuzufügen wird üblicherweise der Container im Konstruktor übergeben. Folgender Code erstellt einen leeren Canvas und fügt ihn zu „parent“ hinzu.

```
new Canvas(Arpent, SWT.NONE);
```

Layout hat verschiedene Optionen, die wichtigste ist die Spaltenanzahl. In dieser Zeile wird „canvas“ ein GridLayout mit 3 unterschiedlich breiten Spalten zugewiesen.

```
canvas.setLayout(new GridLayout(3, false));
```

Um festzulegen wie ein Element in einer Zelle in einem GridLayout angezeigt werden soll weist man dem Element ein LayoutData Objekt zu. Die folgende Programm-Zeile legt fest, dass „button“ horizontal und vertikal zentriert (SWT.CENTER) werden soll. Die Zelle soll vertikal so weit vergrößert werden wie nur möglich (true bei grabExcessVerticalSpace), horizontal nicht (false).

```
button.setLayoutData(new GridData(SWT.CENTER, SWT.CENTER, false, true));
```

Dadurch wird ein Layout möglich, bei dem der gesamte zur Verfügung stehende Bildschirmplatz optimal ausgenutzt wird und kaum fixe Größenangaben mit Pixel nötig sind.

Auf oberste Ebene des Viterbi-Tabs wird ein GridLayout mit nur einer Spalte verwendet, was dazu führt , dass die Elemente einfach untereinander angezeigt werden und den gesamten horizontalen Platz beanspruchen können.

```
setLayout(new GridLayout());
```

Legt das GridLayout für das ViterbiComposite an

```
createHead();
```

Erstellt den Header des Viterbi-Tabs mit Titel und kurzer Beschreibung

```
createInput();
```

Erstellt das Eingabefeld für den Ciphertext

```
createCalculation();
```

Hier können die Parameter des Viterbi Algorithmus eingestellt und die Berechnung gestartet werden.

```
createResult();
```

In diesem Bereich werden die möglichen Plaintexte angezeigt

THREADING

Das ViterbiComposite implementiert das Interface ViterbiObserver, das die Methoden zur Kommunikation mit dem Viterbi-Algorithmus definiert.

Mit der update-Methode informiert der Algorithmus die GUI während der Berechnung über die bereits entschlüsselten Teile, die dann angezeigt werden.

```
public void update(Path path);
```

Wenn die Berechnung abgeschlossen ist wird viterbiFinished() aufgerufen.

```
public void viterbiFinished();
```

Dem Viterbi-Algorithmus wird im Konstruktor eine Referenz auf das ViterbiComposite übergeben.

```
viterbi = new Viterbi(nGramSize, prunningNumber, language,
                        combi, ViterbiComposite.this, cipherString);
isRunning = true;
new Thread(viterbi).start();
```

Damit die Benutzer die Berechnung abbrechen können bietet die Klasse Viterbi die Methode stop() an. Diese setzt ein Flag, das im Algorithmus bei jedem Schleifendurchlauf überprüft wird.