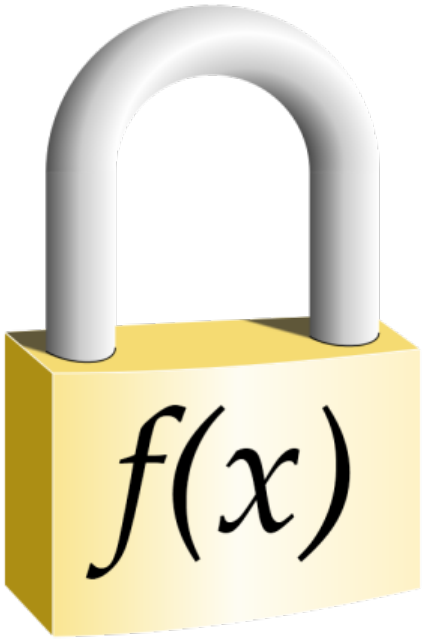
Berner Fachhochschule
Technik und Informatik

Bachelor-Thesis

KryptonIT - Bericht

Password Tresor



[1]

Autoren:
Jan Thomas Liechti
Louis Bernath

Betreuer:
Reto Eric König
Dr. Bernhard Anrig

Experte:
Dr. René Bach

Bern, 23. Januar 2012

Diese Seite wurde absichtlich leer gelassen.

Berner Fachhochschule
Haute école spécialisée bernoise
Technik und Informatik
Technique et informatique

Fachbereich Informatik
Section informatique

Erklärung der Diplomandinnen und Diplomanden
Déclaration des diplômé-e-s

Selbständige Arbeit / Travail autonome

Ich bestätige mit meiner Unterschrift, dass ich meine Bachelor Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.), die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt.

Par ma signature, je confirme avoir effectué mon mémoire de Bachelor de manière autonome. Toutes les sources d'information (littérature spécialisée, discussions avec spécialistes etc.), qui m'ont fortement aidées dans mon travail, sont intégralement mentionnées dans l'annexe de mon mémoire.

Name/Nom, Vorname/Prénom .....

Datum/Date .....

Unterschrift/Signature .....

Dieses Formular ist dem Bachelor Thesis Bericht beizulegen.

Ce formulaire doit être joint au rapport du mémoire de Bachelor.

Diese Seite wurde absichtlich leer gelassen.

Management Summary

Problemstellung Diese Arbeit befasst sich mit der Umsetzung einer kryptographischen Methode mit dem Namen KryptonIT. Die Methode soll als Grundlage für einen Passwort Manager dienen. Der praktische Einsatz der Methode ist weitgehend unerforscht. Aus diesem Grund sollen zwei Umsetzungen („Proof of Concepts“) erstellt werden: Eine Umsetzung, welche in der Lehre eingesetzt werden kann, um die Grundlagen und Funktionsweise grafisch zu vermitteln; eine weitere Umsetzung, mit welcher die Verwendung aus der Sicht eines Benutzers untersucht werden kann. Bei der „Benutzersicht“ soll zusätzlich die Verwendung von Kontext-bezogenen Daten (z.B. ein Wifi-Netzwerk) als Eingaben betrachtet werden, welche in einem modernen Smartphone zur Verfügung stehen. Um einen Passwort-Speicher sowohl in der erklärenden Sicht, als auch in der Benutzersicht untersuchen zu können, soll ein Austausch eines Chiffrates zwischen den beiden Applikationen möglich sein.

Lösung Es wurden zwei Applikationen umgesetzt. Eine Mobile Applikation auf der Basis von Android und eine Web Applikation mit Google Web Toolkit (GWT). Diese Technologien wurden gewählt, da die Umsetzung der KryptonIT Methode als Java Bibliothek bereits zur Verfügung stand.

Mit der Android Applikation können neue Chiffrate erstellt, gespeichert und abgefragt werden. Als Eingaben für die Abfrage wurden Text, der Inhalt eines QR-Barcodes, eine Wifi-Adresse und eine Kompass-Himmelsrichtung umgesetzt. Für den Austausch von Chiffraten wurden Import- und Export-Funktionen realisiert.

Mit der GWT Applikation können ebenfalls neue Chiffrate erstellt, gespeichert, und abgefragt werden. Als Eingaben für die Abfrage wurde ausschliesslich Text umgesetzt, da eine Web Applikation keine Sensoren zur Verfügung hat. Bei der Erstellung und beim Abfragen eines Chiffrates wird dem Benutzer zusätzlich eine grafische Visualisierung und ein Informationsbereich angezeigt. Diese beiden Bereiche zeigen dem Benutzer die Resultate der Berechnungen während der einzelnen Erstellungs- und Abfrage-Schritte eines Chiffrates auf. Des Weiteren hat man die Möglichkeit, ein Chiffrat zu importieren oder exportieren.

Schlussfolgerung Durch grafische Untersuchungen mit der Web Applikation konnte gezeigt werden, dass die ursprünglich eingesetzten Hash-Methoden (z.B. SHA1) für die Verwendung ungeeignet sind. Deshalb musste die Umsetzung der KryptonIT Methode mit einer zusätzlichen Hash-Funktion erweitert werden.

Sowohl für die Android, sowie auch die GWT Applikation konnten, wie in den Anforderungen definiert, umgesetzt werden. Somit stehen die Resultate dieser Arbeit einerseits für die Lehre, andererseits für weitere Untersuchungen der Usability und Sicherheit der KryptonIT Methode zur Verfügung.

Dokumentversionen

Version	Datum	Änderung/Bemerkung
0.1	03.10.2011	Initialisierung des Dokuments.
0.2	31.10.2011	Erster Entwurf Kapitel Analyse.
0.3	15.11.2011	Iteration 1 abgeschlossen.
0.4	08.12.2011	Iteration 2 abgeschlossen.
0.5	29.12.2011	Umstrukturierung der Inhalte.
0.6	06.01.2012	Iteration 3 abgeschlossen.
0.7	16.01.2012	Review und Korrekturen.
1.0	19.01.2012	Veröffentlichung

Inhaltsverzeichnis

1. Einführung

1.1. Ausgangslage

Diese Bachelorarbeit basiert auf der Projektarbeit, welche im Rahmen des Moduls „Projekt 2“ erarbeitet wurde. Dabei wurden die von Reto König et al. [2] aufgestellten theoretischen Grundlagen einer neuen kryptographischen Methode mit dem Namen KryptonIT studiert. Weiter wurde ein „Proof of Concept“ eines Passwort-Manager Programms erstellt, um erste praktische Erfahrungen mit der Methode sammeln zu können.

1.2. Problemstellung

Die kryptographische Methode KryptonIT ist eine neue Verschlüsselungstechnik, welche sich grundlegend von bekannten Verschlüsselungstechniken, wie z.B. die symmetrische oder asymmetrische Verschlüsselung, unterscheidet. Es ist nicht bekannt, dass Applikationen existieren, welche diese Verschlüsselungstechnik nutzen. Deshalb sollen in dieser Arbeit zwei Applikationen die Umsetzbarkeit der neuen kryptographischen Methode als Grundlage für Passwort-Manager überprüfen.

1.3. Ziele der Arbeit

In dieser Bachelorarbeit geht es darum, zwei Passwort-Manager Applikationen zu erstellen, welche die kryptographische Methode KryptonIT verwenden. Die Implementation der Methode steht bereits als Bibliothek zur Verfügung. Das Ziel ist es, diese Bibliothek in einer Web, sowie Mobile Applikation anzuwenden.

Die eine Applikation soll als Web Applikation umgesetzt werden, welche als „erklärende Sicht“ benannt werden kann. Sie erlaubt nicht nur die übliche Funktionalität des Passwort-Managers zu nutzen, sondern sie zeigt dem Benutzer bei der Erstellung eines Chiffrates grafisch die Funktionsweise und einzelnen „Verschlüsselungsschritte“ an und analog, bei der Abfrage des Chiffrates die einzelnen „Entschlüsselungsschritte“ der kryptographischen Methode auf. Durch das Aufzeigen der einzelnen Schritte dient die Applikation als Hilfsmittel, um die kryptographische Methode und deren Sicherheit zu studieren.

Die andere Applikation soll eine Mobile Applikation sein, welche als „Benutzer Sicht“ benannt werden kann. Sie soll dem Benutzer einen möglichst einfachen und intuitiven Zugang zu der kryptographischen Methode bieten, damit das Erstellen, sowie Abfragen eines Chiffrates möglichst einfach gestaltet ist. Um das Chiffrat einfach abzufragen, sollen die Möglichkeiten der Sensoren eines mobilen Gerätes genutzt werden können.

Beide Sichten sollen die Möglichkeit besitzen, ein Chiffrat zu exportieren und wieder zu importieren, um diese auf mehreren Geräten verwenden zu können und zwischen den beiden zu erstellenden Applikationen auszutauschen.

1.4. Stand der Wissenschaft

Um das Gedächtnis des Benutzers zu entlasten und die verschiedensten Passwörter zu speichern, haben sich sogenannte Passwort-Manager durchgesetzt. Diese verwenden ein Haupt-Passwort, (auch Master-Passwort genannt) um die Sammlung von Passwörtern zu verschlüsseln und in einer Datei respektive Datenbank abzulegen. Diese Form der Ablage wird häufig auch verwendet, um unpersönliche Passwörter mit verschiedenen Benutzern auszutauschen (zum Beispiel in einer Firma oder Arbeitsgruppe).

Ein wesentlicher Nachteil dieser Passwort-Manager Systeme ist, dass diese „challenging“ sind (untersucht wurden [3] und [4]). Das heisst, ein Angreifer weiss nach der Eingabe des Master-Passwortes, ob dieses das Richtige ist und kann so durch Probieren in endlicher Zeit (was je nach Stärke des Passwortes und Leistungsfähigkeit des verwendeten Rechners sehr lange ist) den Safe knacken.

Es gibt eine sehr grosse Anzahl von Passwort-Managern auf dem Markt. Hier eine Liste der wichtigsten Vertreter:

- KeyPass, unter der GNU GPL Lizenz [5] stehender Passwort-Manager mit AES oder Twofish Verschlüsselung. [3]
- SplashID, Kostenpflichtiger Passwort-Manager mit AES oder 256-bit Blowfish Verschlüsselung. [6]
- Password-Safe, unter der Artistic License 2.0 [7] stehender Passwort-Manager. [4]

2. Grundlagen

2.1. Kryptographische Methode

Bei Passwortmanagern, wie sie in Kapitel 1.4 erwähnt werden, wird jeweils ein Master Passwort als Schlüssel verwendet, um ein Chiffrat zu entschlüsseln. Bei der KryptonIT Methode hingegen, gibt es mehrere Schlüssel. Mit einem einzelnen Schlüssel kann nur ein Teil des Chiffrates entschlüsselt werden. Deshalb wird auch nicht von *Verschlüsseln* und *Entschlüsseln* sondern von *Erstellen* und *Abfragen* gesprochen. Um die Methode besser erklären zu können und sich von klassischen Verschlüsselungstechicken abzugrenzen, führen wir weitere KryptonIT spezifische Begriffe ein. Ein Chiffrat wird in KryptonIT als *Store* bezeichnet und fasst alle Informationen zusammen (abgesehen von den Schlüsseln), welche benötigt werden, um mit einem Chiffrat zu arbeiten. Der Begriff *Schlüssel* (oder auch *Key*) steht, wie bei der Kryptographie üblich, für die Information, welche benötigt wird, um die Entschlüsselung vorzunehmen. *Geheimnis* (oder *Secret*) wird die Information genannt, welche im Store in verschlüsselter Form enthalten ist.

Erstellung eines Chiffrates

Bei der Erstellung wird im wesentlichen eine Liste von Schlüssel-Geheimnis Paaren definiert. Als Resultat der Verarbeitung durch die Methode erhalten wir ein Chiffrat.

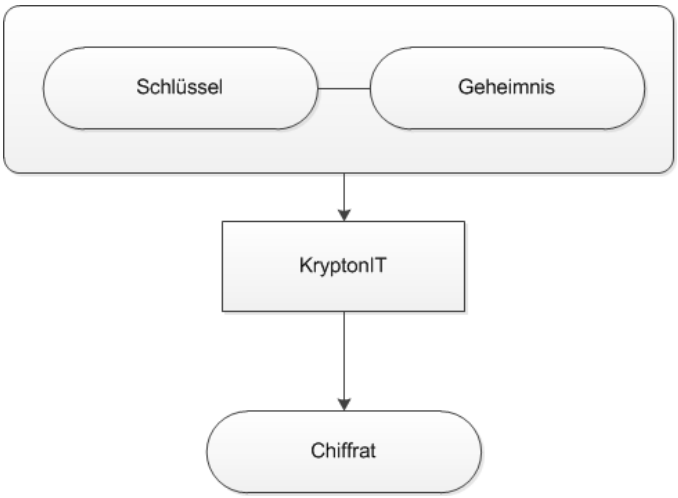


Abbildung 2.1.: Erstellung eines Chiffrates mit KryptonIT.

Abfragen eines Chiffrates

Bei der Abfrage wird das Chiffrat zusammen mit einem Schlüssel an die Methode übergeben. Als Resultat der Abfrage erhalten wir das zum Schlüssel gehörende Geheimnis.

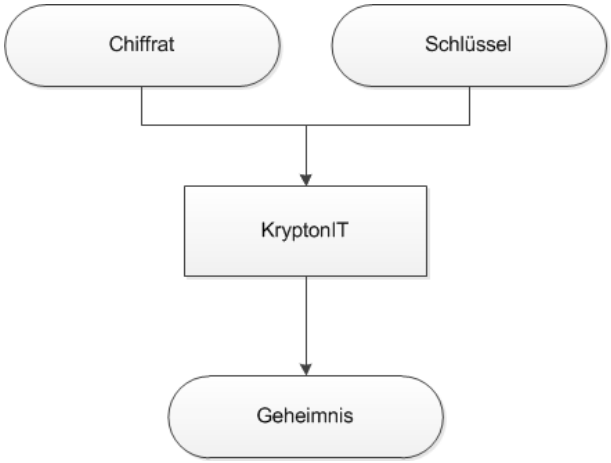


Abbildung 2.2.: Abfrage eines Chiffrates mit KryptonIT.

Eigenschaften der Methode

Der Benutzer wird nicht darauf hingewiesen, ob der verwendete Schlüssel korrekt war oder nicht. Er bekommt in jedem Fall einen Wert zurück („none-challenging“) und muss selbst beurteilen, ob dieser ein Geheimnis ist. Dies ist so, weil der Passwortspeicher ein Polynom ist. Bei einem Polynom können unendlich viele (oder eine sehr grosse Anzahl in einem endlichen Bereich) unterschiedliche Eingaben gemacht werden und es wird immer eine Ausgabe daraus resultieren. Unsere Geheimnisse sind also zwischen sehr vielen anderen „Schein-Geheimnissen“ versteckt. Solange die Schlüssel geheim gehalten werden, kann der ganze KryptonIT Store offen gelegt werden.

Weiter ist in der Methode vorgesehen, dass die verwendeten Schlüssel eine tiefere Entropie als die Geheimnisse haben. Dadurch muss sich der Benutzer nur einfache Schlüssel merken, damit er auf seine Geheimnisse zugreifen kann. Dies macht vor allem dann Sinn, wenn das Geheimnis selbst wieder ein Passwort repräsentiert.

Detailliertere Informationen zu der KryptonIT Methode sind im Anhang A zu finden.

2.2. Anwendungsbereich

Wie im vorigen Abschnitt bereits erwähnt wurde, kann mit KryptonIT ein einfacher Schlüssel verwendet werden, um ein komplexeres Geheimnis zu verschlüsseln. Diese Eigenschaft ist für einen Passwort Manager interessant. Wir stellen uns vor, dass wir uns in ein Computer-System einloggen müssen, welches ein zufälliges und starkes Passwort erfordert. Das System erlaubt drei Login Versuche, bevor der Zugang gesperrt ist. Anstatt dass wir uns das Passwort merken, oder noch schlimmer, im Klartext notieren, speichern wir dieses in einem KryptonIT Store. Das zu merkende Passwort definieren wir als Geheimnis zusammen mit einem kurzen und einfach zu merkenden Schlüssel. Jetzt müssen wir uns nur noch diesen einfachen Schlüssel merken, um an das starke und zufällige Passwort zu gelangen.

3. Zielplattformen

3.1. Web Applikation

Eine Web Applikation [8] ist eine Anwendung, welche auf einem Webserver ausgeführt wird. Die Interaktion mit dem Benutzer erfolgt über das Übertragungsprotokoll HTTP und die Anzeige der Anwendung wird durch einen Webbrowser dargestellt. Dies setzt voraus, dass der Computer des Benutzers (Client) und der Webserver über ein Netzwerk wie z.B. das Internet oder ein Intranet miteinander verbunden sind. Der Vorteil von einer Web Applikation ist, dass diese Plattformunabhängig ist und somit auf jedem Gerät, das einen Webbrowser installiert hat, aufgerufen werden kann.

Rich Internet Application (RIA)

Heutzutage geht der Trend immer mehr in die Richtung, clientseitig mehr Funktionalität einzubauen und somit eine Erweiterung von Web Applikationen (sogenannte Rich Internet Applications [9]) zu erstellen. Unter diesem Begriff werden normalerweise Internetanwendungen verstanden, die eine vielfältige Menge an Interaktionsmöglichkeiten mit ihrer Benutzeroberfläche bieten. Rich Internet Applications beinhalten in der Regel mehr Anwendungslogik als statische Webseiten, die zum Beispiel auf reinem HTML basieren. Durch den Einsatz von Techniken wie z.B. Ajax [10] kann die Performance, sowie Benutzerfreundlichkeit bei der Laufzeit im Gegensatz zu klassischen Webanwendungen spürbar verbessert werden.

Die Vorteile von RIAs sprechen ganz klar dafür, in dieser Arbeit die Realisation der „Erklärenden Sicht“ als Rich Internet Application umzusetzen. Für die Erstellung eines Passwortmanagers, wäre es aus Sicherheitsgründen sogar am sinnvollsten, eine komplett clientseitige Applikation zu erstellen, um den Austausch von sensiblen Informationen zwischen Client und Server über das Netzwerk zu vermeiden. Das bedeutet, die Applikation ist zwar auf einem Webserver abgelegt, um von überall aufrufbar zu sein. Beim ersten Aufruf wird die Applikation jedoch komplett in den Browser geladen und danach wird die Applikation und alle ihre Berechnungen nur noch auf dem Client ausgeführt.

3.2. Mobile Applikation

Als Mobile Applikationen wird die Klasse von Software-Anwendungen bezeichnet, welche auf mobilen Endgeräten zum Einsatz kommt. Wir betrachten für diese Arbeit ausschliesslich die Klasse von Smartphones. Smartphones gibt es schon länger, aber erst in den letzten Jahren hat sich ein regelrechter Boom in diesem Bereich eingestellt. Es gibt eine überschaubare Anzahl von Plattformen, für welche mit Hilfe eines Software Development Kits (SDK) Anwendungen erstellt werden können. Smartphones zeichnen sich typischerweise durch grosse berührungsempfindliche Bildschirme aus, mit welchem das Gerät gesteuert werden kann. Weiter werden in die erwähnte Geräteklasse eine Vielzahl von Sensoren integriert. Die Smartphones sind nicht wie Mobiltelefone für die Nutzung der Telefon-Funktionen optimiert, sondern sollen eine breite Palette von Anwendungen ermöglichen. Sie sind in diesem Sinn ein tragbarer Personal Computer, der für möglichst viele Aufgaben eingesetzt werden kann.

Für die Mobile Applikation sind vor allem die Sensoren, oder allgemeiner formuliert, die Eingabe-Möglichkeiten, von Interesse. Mit diesen möchten wir einen Passwortmanager erstellen, welcher nicht nur die klassische Text-Eingabe erlaubt, um einen Passwort-Tresor zu erstellen und abzufragen, sondern auch die zur Verfügung gestellten Sensoren.

4. Analyse

4.1. Anwendungsfälle

Unser System besteht aus den vier Haupt-Anwendungsfällen „Store erstellen“, „Secret abfragen“, „Store importieren“ und „Store exportieren“.

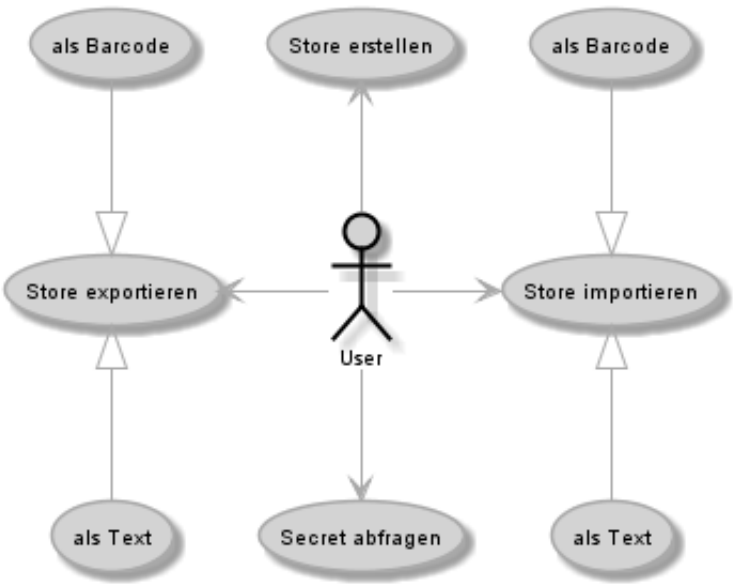


Abbildung 4.1.: Übersicht der Anwendungsfälle

Store erstellen Der Anwendungsfall „Store erstellen“ beschreibt, wie ein neues Chiffrat erstellt wird.

Secret abfragen Der Anwendungsfall „Secret abfragen“ beschreibt, wie ein Secret aus einem bestehenden Store abgefragt werden kann.

Store importieren Der Anwendungsfall „Store importieren“ ist abstrakt und beschreibt, wie ein exportierter Store in das System des Users importiert werden kann. Die Anwendungsfälle „als Barcode“ und „als Text“ sind konkrete Spezialisierungen vom Anwendungsfall „Store importieren“.

Store exportieren Der Anwendungsfall „Store exportieren“ ist abstrakt und beschreibt, wie ein vorhandener Store aus dem System des Users exportiert werden kann. Die Anwendungsfälle „als Barcode“ und „als Text“ sind konkrete Spezialisierungen vom Anwendungsfall „Store exportieren“.

UC_1 - Store erstellen

UC_1	
Titel	Store erstellen
Kurzbeschreibung	Der Akteur will einen neuen Store erstellen. Dafür muss er alle benötigten Informationen im System eingeben.
Vorbedingungen	-
Nachbedingungen	Store wurde erstellt.
Normaler Ablauf	Als Erstes definiert der Akteur die Store spezifischen Parameter für einen neuen Store. Danach gibt er die Schlüssel ein, mit welchen er später die Geheimnisse im Store abfragen kann. Ein Schlüssel kann vom Akteur mit beliebig vielen Teilschlüsseln ergänzt werden, welche mit einer Und-Funktion kombiniert werden. Als letzter Schritt muss für jeden Schlüssel ein Geheimnis eingegeben werden. Anschliessend sind alle Informationen bekannt und der Store wird erstellt.
Sonderfälle	Die Eingabe der Schlüssel kann über verschiedene Eingabe-Typen getätigt werden. Der Benutzer gibt ein Geheimnis ein, welches inkompatibel mit den zu Beginn gewählten Parametern ist.
Spezielle Anforderungen	-

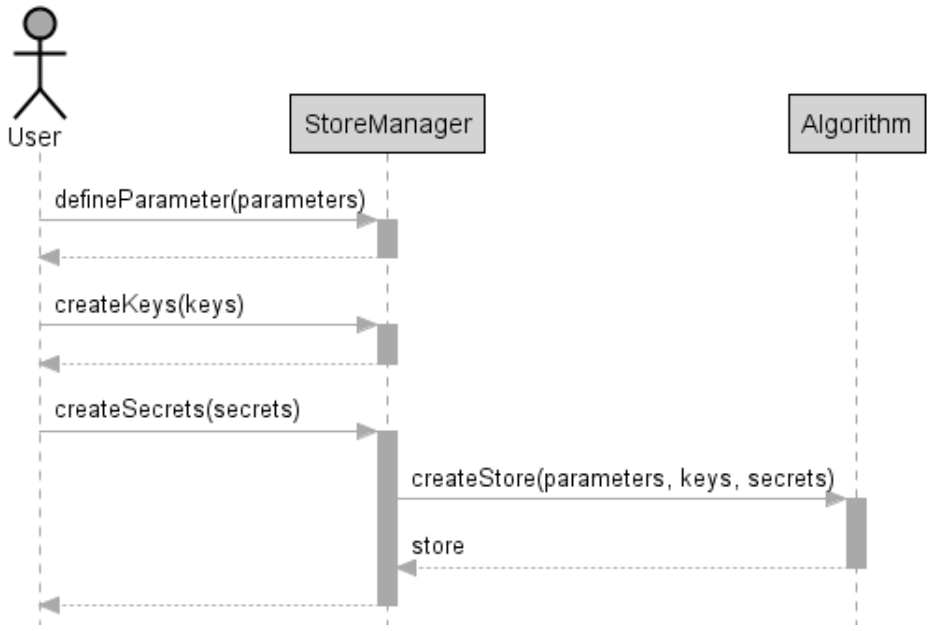


Abbildung 4.2.: Sequenzdiagramm, Store erstellen

UC_2 - Secret aus Store abfragen

UC_2	
Titel	Store abfragen
Kurzbeschreibung	Der Akteur will ein Secret aus einem Store abfragen. Dafür muss er alle benötigten Informationen für eine Abfrage im System eingeben.
Vorbedingungen	Mindestens ein Store ist vorhanden.
Nachbedingungen	Ein Secret wird angezeigt.
Normaler Ablauf	Als Erstes wählt der Akteur einen Store aus. Danach gibt er einen der Schlüssel (kann auch aus mehreren Teilschlüsseln bestehen) ein, mit welchem er vorher den Store erstellt hat. Anschliessend wird das zum eingegebenen Schlüssel passende Geheimnis angezeigt.
Sonderfälle	Die Eingabe der Schlüssel kann über verschiedene Eingabe-Typen getätigt werden.
Spezielle Anforderungen	-

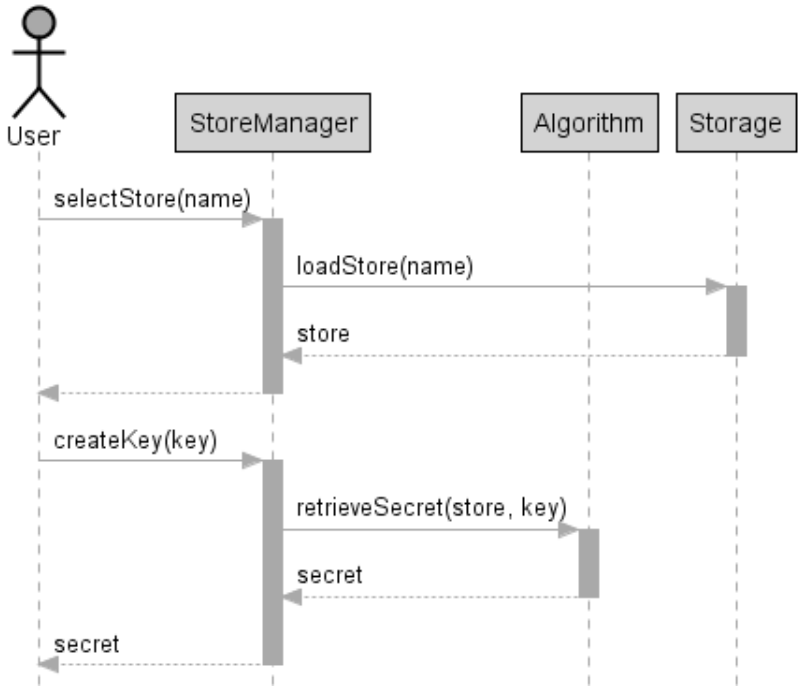


Abbildung 4.3.: Sequenzdiagramm, Secret aus Store abfragen

UC_3 - Store exportieren

UC_3	
Titel	Store exportieren
Kurzbeschreibung	Der Akteur will einen Store für die Verwendung auf einem anderen System oder zur persönlichen Archivierung exportieren.
Vorbedingungen	Mindestens ein Store ist im System vorhanden.
Nachbedingungen	Ein Store wurde im gewählten Exportformat exportiert.
Normaler Ablauf	Als Erstes wählt der Akteur einen Store aus und legt das gewünschte Exportformat fest. Der Store wird in dem gewählten Exportformat bereitgestellt und am Schluss dem User zurückgegeben.
Sonderfälle	-
Spezielle Anforderungen	-

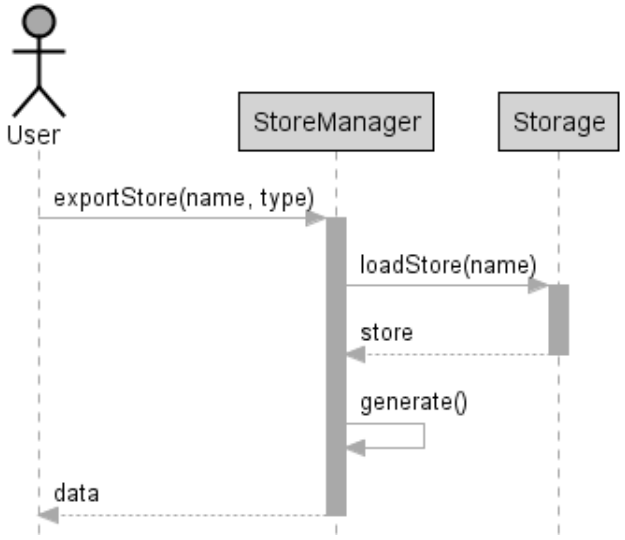


Abbildung 4.4.: Sequenzdiagramm, Store exportieren

UC_3_1 - Store als Text exportieren

UC_3_1	
Titel	Store als Text exportieren
Kurzbeschreibung	Der Akteur will einen Store für die Verwendung auf einem anderen System oder zur persönlichen Archivierung als Text exportieren.
Erweitert	UC_3
Vorbedingungen	Mindestens ein Store ist im System vorhanden.
Nachbedingungen	Der gewählte Store wurde als Text exportiert.
Normaler Ablauf	Der Akteur hat das Exportformat „Text“ gewählt. Der Text kann danach abgespeichert werden.
Sonderfälle	-
Spezielle Anforderungen	-

UC_3_2 - Store als Barcode exportieren

UC_3_2	
Titel	Store als Barcode exportieren
Kurzbeschreibung	Der Akteur will einen Barcode vom Store generieren, um diesen direkt mit einem anderen System auszutauschen oder abzuspeichern.
Erweitert	UC_3
Vorbedingungen	Mindestens ein Store ist vorhanden.
Nachbedingungen	Der Barcode des gewählten Store's wird angezeigt.
Normaler Ablauf	Der Akteur hat das Exportformat „Barcode“ gewählt. Danach wird dem Akteur ein Barcode angezeigt. Zusätzlich hat er die Möglichkeit diesen abzuspeichern.
Sonderfälle	-
Spezielle Anforderungen	-

UC_4 - Store importieren

UC_4	
Titel	Store importieren
Kurzbeschreibung	Der Akteur will einen exportierten Store ins System importieren.
Vorbedingungen	Der zu importierende Store hat ein gültiges Format.
Nachbedingungen	Store wurde ins System importiert.
Normaler Ablauf	Als Erstes wählt der Akteur einen vorher exportierten Store aus, welchen er importieren will. Dieser wird vom System validiert und im System abgespeichert.
Sonderfälle	Wird der zu importierende Store vom System als ungültig erkannt, wird dem Akteur eine entsprechende Meldung angezeigt und das System bricht den Vorgang ab.
Spezielle Anforderungen	-

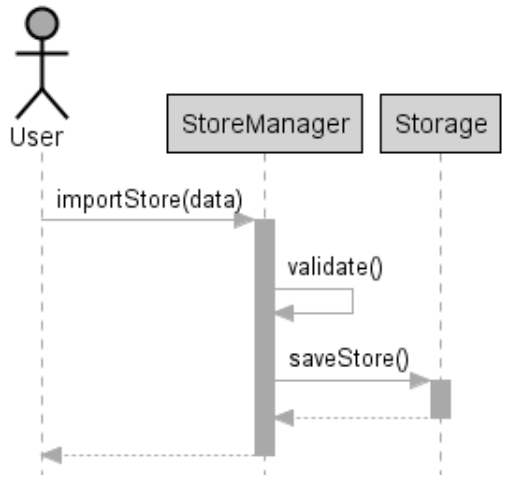


Abbildung 4.5.: Sequenzdiagramm, Store importieren

UC_4_1 - Store aus Text importieren

UC_4_1	
Titel	Store aus Text importieren
Kurzbeschreibung	Der Akteur will einen exportierten Store aus einem Text ins System laden.
Erweitert	UC_4
Vorbedingungen	Der zu importierende Store hat ein gültiges Format und ist als Text vorhanden.
Nachbedingungen	Store wurde ins System importiert.
Normaler Ablauf	Der Akteur wählt einen als Text exportierten Store aus. Dieser wird vom System validiert und abgespeichert.
Sonderfälle	Wird der zu importierende Store vom System als ungültig erkannt, wird dem Akteur eine entsprechende Meldung angezeigt und das System bricht den Vorgang ab.
Spezielle Anforderungen	-

UC_4_2 - Store aus Barcode importieren

UC_4_2	
Titel	Store aus Barcode importieren
Kurzbeschreibung	Der Akteur will einen Store aus einem Barcode ins System importieren.
Erweitert	UC_4
Vorbedingungen	Der exportierte Store ist gültig und als Barcode vorhanden.
Nachbedingungen	Store wurde ins System importiert.
Normaler Ablauf	Der Akteur liest einen Store mit dem Barcode-Leser ein. Die Barcode-Daten werden ans System weitergegeben, welches diese validiert und im System abspeichert.
Sonderfälle	Wird die vom Barcode-Leser ans System weitergegebene Information als ungültiges Format erkannt, wird dem Akteur eine entsprechende Meldung angezeigt und das System bricht den Vorgang ab.
Spezielle Anforderungen	Dieser Anwendungsfall kann nur auf einem Gerät mit Barcode-Leser ausgeführt werden.

4.2. Verwendung der KryptonIT Bibliothek

Um die Kryptonit Methode anzuwenden, wurde von unserem Betreuer Reto König eine Bibliothek zur Verfügung gestellt. Diese Bibliothek ist als Java Projekt vorhanden und erlaubt es, Chiffrate zu erstellen und abzufragen. Hier sind die verschiedenen Parameter aufgelistet, welche nötig sind, um ein Chiffrat zu erstellen.

- Key : Schlüssel, welcher zusammen mit einem Geheimnis zu einem Punkt im finiten Körper (siehe A.4) führt.
- Secret : Geheimnis, welches im Chiffrat „versteckt“ ist.
- Secret Space : Theoretische Anzahl der möglichen Geheimnisse und zugleich das grösstmögliche Geheimnis im Chiffrat. Wird bei der Erstellung angegeben.
- Security Order (Sicherheitsparameter 1): Beeinflusst die Sicherheit des Chiffrates, in dem Polynome höheren Grades erzeugt werden.
- Secret Space Factor (Sicherheitsparameter 2): wird zusammen mit dem Secret Space multipliziert, um den Wert P zu vergrössern. Wird bei der Erstellung angegeben.
- Hash Method : Die von der Kappa-Funktion verwendete Hashfunktion. Hier stehen eine der drei Funktionen MD5, SHA1, SHA256 oder EXP zur Verfügung. Wird bei der Erstellung angegeben.
- Number of Random Points : Anzahl der zusätzlich zu den eingegebenen Key-Secret Paaren zufällig generierten Punkte. Wird bei der Erstellung angegeben.

Ablauf der Erstellung:

1. Definieren des Secret Space inkl. Security Order. Dabei wird die Grösse des finiten Körpers P berechnet (siehe Kapitel 4.5).
2. Abbilden der Keys auf die X-Achse des finiten Körpers mittels der Kappa Funktion. Bei der Abbildung wird ein zufälliger Offset gewählt und sichergestellt, dass mit diesem keine Kollisionen der X-Werte auftreten.
3. Abbilden der Secrets auf die Y-Achse des finiten Körpers mit der Sigma Funktion.
4. Bilden der Stützpunkte mit den X und den dazugehörigen Y Werten.
5. Erstellen des Stores mit Hilfe der Polynominterpolation.

Bei der Erstellung eines Chiffrats werden, wie oben erwähnt, folgende Werte berechnet:

- P : Die Grösse des finiten Körpers.
- Offset : Eine zufällig gewählte Zahl, welche ein kollisionsfreies Abbilden der Keys in den finiten Körper erlaubt.

4.3. Schnittstelle auf die KryptonIT Bibliothek

Die Web, sowie die Mobile Anwendung sollen über eine gemeinsame Schnittstelle auf die Bibliothek zugreifen. Die Schnittstelle haben wir Fassade genannt, weil es sich an einem Entwurfsmuster der objektorientierten Softwareentwicklung anlehnt. Mit der Fassade werden zwei Ziele verfolgt. Einerseits soll die Nutzung der KryptonIT Bibliothek vereinfacht werden, indem wir eine zusätzliche Abstraktionsebene einführen, welche genau unseren Anforderungen entspricht. Andererseits dient unsere Fassade als Behälter, um zusätzliche Funktionalität zur Verfügung zu stellen, welche von beiden Applikationen verwendet werden kann, wie z.B. ein gemeinsames Import-/Export-Datenformat der Chiffrate.

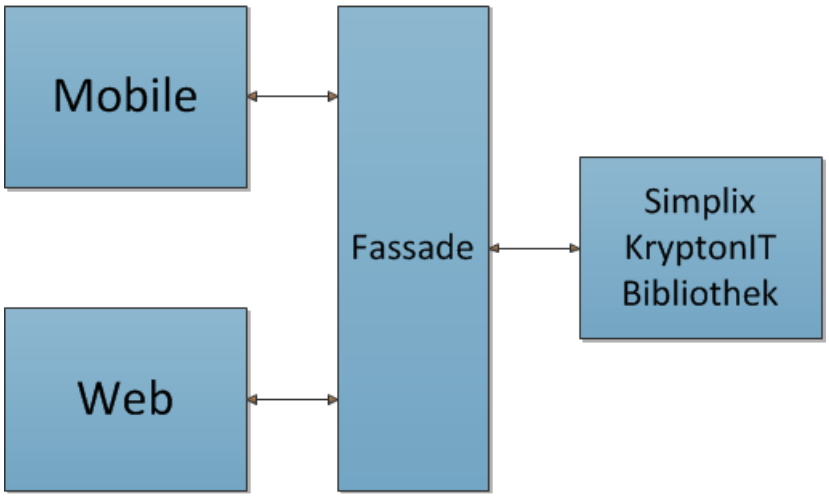


Abbildung 4.6.: Beziehung zwischen der Web und Mobile Applikation mit der KryptonIT Library.

4.4. Persistierung eines Stores

Für die Persisitierung wird das Daten-Modell der KryptonIT Fassade verwendet. Damit ist es mit den serialisierungs und deserialisierungs Mechanismen von Java möglich, einen Store oder Teilresultate in einem Datenformat abzulegen oder für die geforderte Export/Import Funktionalität auszutauschen. Das JSON Datenformat bietet sich besonders an, da kleinere Formate eines Stores möglich sind (siehe Beispiele im Anhang B.2).

4.5. Wahl des Secret Space

Die Grösse des Secret Space hängt stark damit ab, ob nur Zahlen oder alphanumerische Geheimnisse (im späteren Verlauf auch Text Geheimnisse genannt) gespeichert werden sollen. Bei Text Geheimnissen kommt erschwerend hinzu, dass bei einer Abfrage eines Stores sofort ersichtlich ist, ob es sich dabei um ein Geheimnis handelt oder nicht. Dies kommt daher, dass in üblichen Text Encoding auch Steuerzeichen oder spezielle Sonderzeichen enthalten sind, welche der Benutzer nur schwer als Symbol in einem Passwort verwenden kann. Deshalb kann keines der im Anhang B.1 erwähnten Text Encodings verwendet werden. Ausser es handelt sich bei dem Geheimnis des Benutzers um eine rein zufällig gewählte Zeichenfolge, welche möglichst normal verteilt in den kompletten Raum des betreffenden Encodings fällt. Deshalb muss

das Encoding vom Benutzer möglichst genau auf seine Bedürfnisse abgestimmt werden können. D.h. der Benutzer sollte wählen können, ob er den ganzen UTF-16 Zeichen-Raum, nur Zeichen von A bis z, Zeichen von A bis z und Sonderzeichen, nur Zahlen, etc. verwenden möchte. Ähnlich wie bei einem Passwort-Generator das zu verwendende Alphabet für die zufälligen Passwörter definiert wird, muss bei der Wahl des Secret Space berücksichtigt werden, welches Alphabet verwendet werden soll.

Um eine möglichst lückenlose Abbildung vom Secret Space in den Passwortzeichen-Raum zu erhalten, werden nur eine eingeschränkte Anzahl von Zeichen als Geheimnisse zugelassen. Die gewählte Zeichenmenge wurde inspiriert aus dem Passwortgenerator von KeePass [11] ausgewählt (siehe auch Anhang B.1).

4.6. Schlüssel Kombinationen

Um mehrere Teil-Schlüssel in einem Schlüssel zu kombinieren, verwenden wir die Methode aus dem Vorprojekt [12]. Folgende Bedingungen stellen wir an eine Schlüssel Kombination:

- Die Reihenfolge, in welcher die Kombination vorgenommen wird, darf keinen Einfluss auf das Resultat haben.
- Ein Schlüssel darf nicht auf 0 zusammenfallen.

In der oben erwähnten Methode, kann der zweite genannte Punkt nicht garantiert werden. Um dies zu verhindern, müssen die Web und Mobile Applikation selbst validieren, dass die kombinierten Werte eindeutig sind.

4.7. Nutzung von Hash-Methoden mit grossen Zahlen

Im Laufe der Entwicklung der Web Applikation konnte mit Hilfe der Visualisierung festgestellt werden, dass bei der Abbildung von Schlüsseln in den Geheimnis-Raum grösser als 2^{128} die Berechneten x-Werte (innerhalb der Kappa-Funktion) immer sehr nahe bei Null waren oder fast die Grösse des Geheimnis-Raums hatten. Daraufhin haben wir empirisch festgestellt, dass die KryptonIT Bibliothek mit den Hash-Methoden MD5, SHA1 und SHA256 nicht verwendet werden können. Deshalb haben wir die Hilfe unseres Betreuers, Reto König, in Anspruch genommen. Mit seiner Unterstützung haben wir eine weitere Hash-Methode in die Bibliothek eingebaut. Die Methode nennt sich Modulare Exponentiation [13] und wird in der Kryptographie verwendet.

4.8. Web Applikation

4.8.1. Google Web Toolkit

Google Web Toolkit [14][15] ist ein Framework zur Entwicklung von Webanwendungen. Es wurde von Google als freie Software unter den Bedingungen von Version 2.0 der Apache-Lizenz veröffentlicht. Seine Besonderheit ist ein Java nach JavaScript Compiler, sodass nahezu die gesamte Entwicklung von Client und Server auf der Basis von Java realisiert werden kann. Weiter ist GWT mit einem XML-Parser, Internationalisierungs-Unterstützung, einer Schnittstelle für Remote Procedure Calls, Integration von JUnit [16] und einem kleinen Widget-Paket zur Gestaltung der graphischen Oberfläche ausgestattet. Diese kann dabei ähnlich wie mit Swing [17] erstellt werden. Was GWT im Wesentlichen von den meisten anderen Frameworks dieser Art unterscheidet, ist die Tatsache, dass auch der clientseitige Code komplett in Java erstellt werden kann. Dies bringt erhebliche Vorteile in der Entwicklung mit sich, weil bewährte Entwicklungsumgebungen benutzt werden können. Des Weiteren ist ein interner Anwendungsserver (z.B. Apache Tomcat [18]) enthalten, der beim Entwickeln zum Einsatz kommt.

Die Eigenschaften von GWT erfüllen genau die im Kapitel 3.1 erwähnten Bedingungen, um einen sicheren Passwortmanager als RIA erstellen zu können. Da Java Kenntnisse bereits vorhanden sind und die Krypton-IT Bibliothek als Java-Implementierung vorliegt, wurde entschieden, keine weiteren Recherchen über andere Frameworks durchzuführen und GWT für die Umsetzung der erklärenden Sicht einzusetzen.

4.8.2. Lauffähigkeit der KryptonIT Bibliothek mit Google Web Toolkit

Um die Lauffähigkeit der KryptonIT Bibliothek mit Google Web Toolkit zu prüfen, wurde ein Testprojekt erstellt und die KryptonIT Bibliothek eingebunden. Als wichtigster Punkt war zu prüfen, ob die ganze Applikation in JavaScript kompiliert werden kann, um eine komplett clientseitige Applikation erstellen zu können. Es wurde festgestellt, dass zusätzliche Bibliotheken von Google Web Toolkit benötigt werden, welche aber in der aktuellen Version von Google Web Toolkit nicht vorhanden sind (siehe Anhang B.3.1). Anhand der Erkenntnisse, dass die KryptonIT Bibliothek im Moment nicht für den clientseitigen Gebrauch kompiliert werden kann, wurde entschieden, die KryptonIT Bibliothek serverseitig laufen zu lassen und somit eine Client-Server Applikation zu erstellen. Wenn die KryptonIT Bibliothek serverseitig ausgeführt wird, funktioniert alles einwandfrei. Der Nachteil davon ist aber, dass durch die Client-Server Kommunikation sensible Daten übers Netz übertragen werden, was ein Sicherheitsrisiko darstellen kann (siehe Kapitel 4.8.7).

4.8.3. Aufbau der Applikation

Da mit Google Web Toolkit Webanwendungen erstellt werden, kann der Code von der Anwendung entweder clientseitig oder serverseitig ausgeführt werden. Soll der in Java geschriebene Code clientseitig oder besser gesagt im Browser des Users ausgeführt werden, wird dieser beim Kompilieren komplett in JavaScript übersetzt. Für diese Anwendung wäre es naheliegend, alles clientseitig ausführen zu lassen. Weil dies aber leider im Moment mit den verfügbaren GWT-Klassen nicht möglich ist (siehe Kapitel 4.8.2), wird eine Client-Server Applikation umgesetzt.

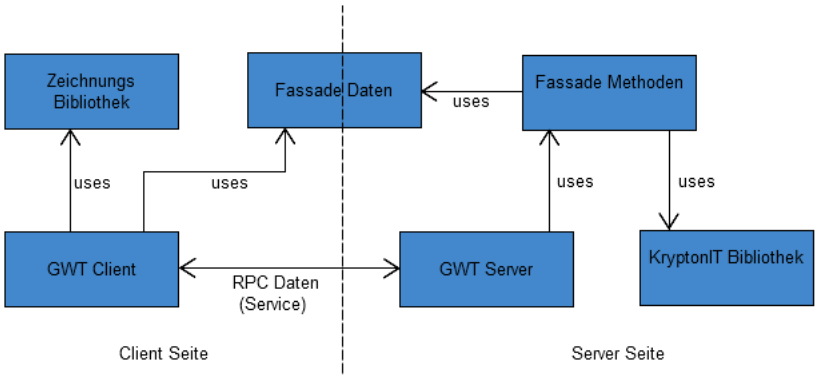


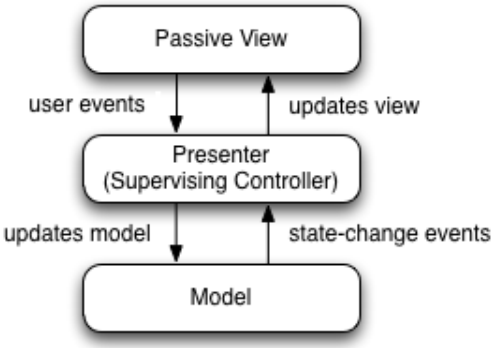
Abbildung 4.7.: Aufbau der Client-Server Applikation

Auf der Client-Seite werden alle Eingaben gemacht und alle Darstellungen gezeichnet. Weiter wird clientseitig das Modell der KryptonIT Fassade (siehe Kapitel 6.1) als Modul eingebunden und die benötigten Interfaces für den Service, welcher für den Zugriff auf die serverseitigen Methoden zuständig ist. Die Fassade Modell-Klassen stellen die Objekte dar, welche zur Übermittlung zwischen Client und Server im Service verwendet werden.

Auf der Server-Seite wird die KryptonIT Bibliothek ausgeführt, welche dem Service die bereits erwähnten Methoden zur Verfügung stellt. Sie empfängt die Parameter, um einen Store zu erstellen oder einen Store abzufragen und gibt die einzelnen Werte für die Darstellung dem Client über den Service wieder zurück.

Model-View-Presenter Entwurfsmuster (MVP)

Das MVP [19] ist ein Entwurfsmuster in der Softwareentwicklung, welches aus dem Model-View-Controller (MVC) [20] hervorgegangen ist. Es beschreibt einen neuartigen Ansatz, um das Modell und die Ansicht komplett voneinander zu trennen und über einen Präsentator zu verbinden. Dabei steht neben einer deutlich verbesserten Testbarkeit auch die strengere Trennung der einzelnen Komponenten im Gegensatz zu MVC im Vordergrund.



[21]

Abbildung 4.8.: Aufbau der Applikation

Dieser neue Ansatz klingt sehr interessant. Deshalb wurde entschieden, das MVP Entwurfswuster beim Aufbau der Applikation anzuwenden, um die beschriebenen Vorteile ausnutzen zu können.

4.8.4. Unterschiedliche Ansichten

Dem Benutzer werden beim Erstellen, sowie beim Abfragen eines Stores unterschiedliche Ansichten angeboten. Die Ansichten unterscheiden sich in der Anzahl der preisgegebenen Informationen über die ausgeführten Berechnungen und erhaltenen Zwischenresultate.

Ansichten beim Erstellen eines Stores

Beim Erstellen eines Stores kann der Benutzer zwei unterschiedliche Ansichten wählen. Die eine Ansicht wird Benutzer Ansicht genannt. Sie stellt dem Benutzer kompakt alle Eingabefelder, Selektoren und Knöpfe zur Verfügung, welche benötigt werden, um einen Store mit den gewünschten Schlüsseln und zugehörigen Geheimnissen erstellen zu können. Die zweite Ansicht wird erweiterte Ansicht genannt. In der erweiterten Ansicht wird zusätzlich eine grafische Ausgabe in Form eines Koordinatensystems und weiteren Achsen angezeigt. Weiter wird ein Informationsbereich eingeblendet, welcher dem Benutzer die Ergebnisse der durchgeführten Berechnungen während den einzelnen Erstellungsschritten aufzeigt.

Ansichten beim Abfragen eines Stores

Um unterschiedliche Aspekte der Sicherheit der KryptonIT Bibliothek aufzeigen zu können, kann beim Abfragen eines Geheimnisses aus dem Store zwischen 5 unterschiedlichen Ansichten gewechselt werden. Die Sichten unterscheiden sich in der Menge der Informationen, welche vom System bekannt gegeben werden. Die erste Ansicht ist die Benutzer Sicht. Sie stellt dem Benutzer kompakt alle Eingabefelder, Selektoren und Knöpfe zur Verfügung, welche benötigt werden, um ein Geheimnis aus einem Store abfragen zu können. In den vier weiteren Ansichten wird zusätzlich eine grafische Ausgabe in Form eines Koordinatensystems und weiteren Achsen angezeigt. Weiter wird ein Informationsbereich eingeblendet, welcher dem Benutzer die Ergebnisse der durchgeführten Berechnungen während den einzelnen Abfrageschritten aufzeigt. Die vier weiteren Ansichten stellen unterschiedliche Angreifer-Modelle dar. In der Ansicht des ersten Angreifer-Typs, wird dem Benutzer am wenigsten Information preisgegeben und in der Ansicht des vierten Angreifer-Typs werden alle Informationen des Systems ausgegeben.

Angreifer Typ 1 (kennt am wenigsten Informationen)

Der Angreifer vom Typ 1 sieht den Store und das resultierende Geheimnis. Das bedeutet, in der graphischen Ausgabe wird nur die Geheimnis-Achse eingeblendet und im Informationsbereich werden nur die Informationen des verwendeten Stores und der Geheimnis-Wert angezeigt.

Angreifer Typ 2

Der Angreifer vom Typ 2 sieht zusätzlich den verwendeten y-Wert. Das bedeutet, in der graphischen Ausgabe wird zusätzlich die y-Achse eingeblendet und im Informationsbereich wird der y-Wert angezeigt.

Angreifer Typ 3

Der Angreifer vom Typ 3 sieht zusätzlich den verwendeten x-Wert. Das bedeutet, in der graphischen Ausgabe wird zusätzlich die x-Achse eingeblendet und im Informationsbereich wird der x-Wert angezeigt.

Angreifer Typ 4

Der Angreifer vom Typ 4 sieht zusätzlich den verwendeten Schlüssel-Wert. Das bedeutet, in der graphischen Ausgabe wird zusätzlich die Schlüssel-Achse eingeblendet und im Informationsbereich wird der Schlüssel-Wert angezeigt.

4.8.5. Hilfe Texte

Damit der Benutzer bei Bedarf detaillierte Informationen abfragen kann, werden in der Applikation 12 Informations-Icons angezeigt. Klickt der Benutzer auf ein solches Informations-Icon, wird ein Fenster geöffnet, welches die Informationen zum gewünschten Schritt anzeigt. Die Fenster bleiben solange geöffnet, bis sie vom Benutzer wieder geschlossen werden.

Hier werden die einzelnen Informationen aufgelistet und kurz erläutert:

Store erstellen Seite

- Ansicht : Erklärt die unterschiedlichen Anzeige-Ansichten beim Erstellen eines Stores.
- Finiter Körper : Erklärt die einzelnen Eingabeparameter des Schritts Finiter Körper.
- Schlüsseleingabe : Erklärt die einzelnen Eingabeparameter des Schritts Schlüsseleingabe.
- Geheimniseingabe : Erklärt die einzelnen Eingabeparameter des Schritts Geheimniseingabe.
- Polynominterpolation : Erklärt die einzelnen Eingabeparameter des Schritts Polynominterpolation.
- Store speichern : Erklärt die einzelnen Eingabeparameter des Schritts Store speichern.

Store abfragen Seite

- Ansicht : Erklärt die unterschiedlichen Anzeige-Ansichten beim Abfragen eines Stores.
- Store laden : Erklärt die einzelnen Eingabeparameter des Schritts Store laden.
- Store abfragen : Erklärt die einzelnen Eingabeparameter des Schritts Store abfragen.

Import/Export Seite

- Store importieren : Erklärt die einzelnen Eingabeparameter beim Importieren eines Stores.
- Store exportieren : Erklärt die einzelnen Eingabeparameter beim Exportieren eines Stores.
- Store löschen : Erklärt die einzelnen Eingabeparameter beim Löschen eines Stores.

4.8.6. Grafische Darstellung mit GWT

Um den Ablauf beim Erstellen oder Abfragen eines Stores grafisch aufzeigen zu können, eignet sich am Besten die Darstellung in einem Koordinatensystem. Ein Schlüssel-Geheimnis Paar stellt im Koordinatensystem die x- und y-Werte dar. Zusätzlich werden weitere Werte-Achsen benötigt, um die Schlüssel und Geheimnisse vor den Transformationen anzeigen zu können. Eine solche Darstellung muss selber, durch Zeichnen einer Grafik, erstellt werden.

Zuerst wurde untersucht, was es für Möglichkeiten gibt, mit Goolge Web Toolkit eine grafische Ausgabe zu erstellen. Drei Bibliotheken wurden genauer untersucht.

- Canvas - Bibliothek für Bitmap-Grafiken [22]
- lib-gwt-svg - Bibliothek für SVG-Grafiken [23][24]
- gwt-graphics - Bibliothek für SVG-Grafiken [25][26]

Die beiden Bibliotheken, welche SVG Grafiken erstellen, wurden bevorzugt, weil diese Vektor basiert sind. Somit schied Canvas früh aus. Für beide Varianten wurde ein Prototyp erstellt. Beide Ergebnisse waren fast identisch. Da die Bibliothek gwt-graphics aber intuitiver zu Benutzen ist, wurde entschieden, diese im Projekt einzusetzen.

4.8.7. Sicherheit des Systems

Weil keine Client Applikation, sondern eine Client-Server Applikation umgesetzt wird und der Fokus auf der Erklärung und Visualisierung der Funktionalität der KryptonIT-Bibliothek liegt, wird bewusst auf den Aspekt der Sicherheit verzichtet und auch keine Verschlüsselung der Client-Server Kommunikation durchgeführt. Die GWT Applikation, wie sie in diesem Projekt umgesetzt wird, soll also nicht wie man sich denkt, als Passwort-Manager genutzt werden, sondern nur als Erklärungsinstrument, wie dieser funktioniert und aufgebaut ist, dienen.

Um die Web Applikation sicher machen zu können und sie dann zusätzlich auch in der eigentlich dafür gedachten Funktion als Passwort-Manager nutzen zu können, müsste sie komplett clientseitig im Browser des Benutzers laufen. Somit würden keine sensiblen Daten mehr übers Netz geschickt, da keine Client-Server Kommunikation mehr stattfindet. (siehe Anhang B.3.3)

4.8.8. Speichern und Laden eines Stores

Bei einer Web Applikation, gibt es grundsätzlich zwei verschiedene Möglichkeiten, Daten zu speichern. Bei der ersten Variante werden die Daten auf dem Server gespeichert, auf welchem die Applikation abgelegt ist. Dies benötigt ein Benutzerlogin, um sich eindeutig zu identifizieren. Bei der zweiten Variante werden die Daten auf dem Client im Browser gespeichert, von welchem die Applikation aufgerufen wird. Das Speichern auf dem Client wird bevorzugt, da es angedacht ist, zu einem späteren Zeitpunkt, eine komplett clientseitige Web Applikation zu erstellen. Als Erstes wurde untersucht, ob sich HTTP-Cookies [27] als Speicher für einen Store eignen würden. Leider sind HTTP-Cookies auf 4 kB begrenzt, was für einen Store nicht ausreicht. Neuerdings gibt es eine weitere Möglichkeit, Daten auf dem Client im Browser abzulegen, die

Verwendung eines Local Storage [28]. Ein Local Storage ist auf 5 MB begrenzt (im Internet Explorer 10 MB), was im Vergleich zu Cookies erheblich mehr Speicherplatz bietet. Nach der Erkenntnis, dass GWT standardmässig die Nutzung von eines Local Storages unterstützt, wurde entschieden, Local Storage zu verwenden, um einen Store speichern zu können. Die zu speichernden Klassen werden mit Hilfe von JSON [29] serialisiert (siehe Kapitel 4.4) und danach im Browser abgelegt.

4.8.9. Importieren eines Stores

Eine Web Applikation kann clientseitig nur auf Files zugreifen, welche auf demselben Webserver abgelegt sind, auf welchem die Webapplikation läuft. In Anbetracht, dass die Applikation zu einem späteren Zeitpunkt ganz clientseitig umgesetzt wird, entschied man sich auch hier, eine clientseitige Lösung umzusetzen. Es wird ein mehrzeiliges Textfeld angeboten, in welches ein zuvor exportierter Store im JSON-Format (Text) eingefügt werden kann. Dieser wird eingelesen, auf inhaltliche Gültigkeit überprüft und importiert. Der Store ist nun genau gleich wie beim Speichern eines Stores (siehe Kapitel 4.8.8) im Local Storage des Browsers abgelegt.

4.8.10. Exportieren eines Stores

Auch beim Exportieren eines Stores wird eine Client seitige Lösung bevorzugt. Deshalb, kann analog dem Importieren (siehe Kapitel 4.8.9), ein Store als JSON-String (Text) exportiert werden. Dieser wird in einem mehrzeiligen Textfeld ausgegeben, woraus er manuell kopiert und z.B. auf dem Computer abgespeichert werden kann. Als zweite Exportmöglichkeit wird das Exportieren eines Stores als „Quick Respone Code“ angeboten. Der angezeigte QR-Code kann nun von einem Mobilen Gerät eingelesen werden. (siehe Kapitel 4.9.12)

4.8.11. Slider

Für die Auswahl einiger Werte beim Erzeugen des Finiten Körpers eignet sich am Besten ein Slider-Auswahlwerkzeug, da es dem Benutzer einen eingeschränkten Wertebereich zur Verfügung stellt, in welchem er einen Wert durch verschieben des Sliders auswählen kann. In GWT wird standardmässig leider kein Slider-Widget angeboten. Es musste sehr lange (ca. 2 Tage) im Internet recherchiert werden, bis eine Alternative gefunden wurde, welche mit der aktuellen GWT Version funktionierte. [30]

4.8.12. GWT JUnit Tests

Von GWT wird eine auf JUnit basierende Testinfrastruktur zur Verfügung gestellt, welche erlaubt, die Tests in einem Browser-Kontext laufen zu lassen. [31] Die Ausführung der Tests kann bequem in Eclipse mit Hilfe des Google Plugin für Eclipse, über die Kommandozeile oder im „manuellen Test-Modus“ laufen gelassen werden. Die Testklassen werden im selben Projekt in einem separaten Source-Ordner erstellt. [31][21]

4.9. Mobile Applikation

4.9.1. Android

Als Mobile Plattform haben wir Android gewählt, weil die Applikations-Entwicklung mit Java angeboten wird und so die KryptonIT Bibliothek eins zu eins verwendet werden kann. Zudem ist die Plattform sehr offen gestaltet, was den Zugang zu den Sensoren und anderen Hardware-Bausteinen erleichtert. Android bezeichnet ein auf dem Linux-Kernel basierendes Betriebssystem, welches von Google erstellt und verwaltet wird. Das Betriebssystem selbst ist Quelloffen und kann von jedem verwendet oder modifiziert werden. Aus der Sicht eines Anwenders ist Android eine Plattform und ein Ökosystem für Smartphones (und mittlerweile auch für Tablets). Der Zugang zu dem Ökosystem ist mit dem reinen quelloffenen Betriebssystem nicht möglich, da diesem der Zugang zu dem Android Market fehlen. Der Market bezeichnet die Schnittstelle, in Form einer Anwendung, wo der Benutzer neue Anwendungen auf sein Gerät herunterladen und installieren kann. Da der Market aber nicht quelloffen ist, werden nur von Google zertifizierte Geräte mit dem Market ausgestattet.

4.9.2. KryptonIT mit Android

Um die Lauffähigkeit der KryptonIT Bibliothek zu überprüfen, wurde eine Android Applikation erstellt. Diese erstellt ein Chiffrat mit fest definierten Parametern und entschlüsselt die Werte danach gleich wieder. Dabei sind keine Probleme aufgetreten. Für die Mobile Applikation kann die KryptonIT Bibliothek ohne Anpassungen verwendet werden.

4.9.3. Android Versionen und API Levels

Jede Version von Android ist ein API Level zugewiesen. Dieser API Level bestimmt den zur Verfügung stehende Funktionsumfang des Android Frameworks. Die Framework Versionen sind grundsätzlich Vorwärtskompatibel.

Jede Applikation muss in ihrem Manifest definieren, welchen API Level sie im Minimum verwendet. Wird ein Gerät mit einer tieferen Version verwendet, lässt sich die Applikation nicht installieren.

Bei der Wahl des API Levels stehen technische Rahmenbedingungen im Vordergrund. Möchte man z.B. die NFC Möglichkeiten von Android verwenden, muss man mindestens API Level 9 verwenden. Neben den technischen Überlegungen ist auch die Verbreitung der verschiedenen Android Versionen ein Faktor, dem man Beachtung schenken sollte. Das Android Ökosystem ist im Vergleich mit anderen mobilen Plattformen einer starken Fragmentierung unterworfen. Dies ist hauptsächlich darauf zurückzuführen, dass es viele verschiedene Hersteller von Android Geräten gibt, für welche aufgrund technischer und wirtschaftlicher Überlegungen keine Aktualisierungen auf neuere Android Versionen angeboten werden.

4.9.4. Aufbau der Applikation

Das Android Framework bietet vier verschiedene Arten von Komponenten an, um Applikationen zu erstellen [32]. Diesem Konzept wollen wir so nah wie möglich folgen, damit KryptonIT für Android als flexible und wiederverwendbare Lösung konzipiert ist.

Für Benutzeroberflächen stellt Android sog. Activities bereit. Für jeden „Bildschirm“ wird eine Activity entwickelt. Zwischen den Activities kann mit Intents navigiert werden, welche auch die Über- und Weitergabe von Daten zwischen den Activities ermöglicht.

Das Speichern und Laden der Stores soll durch einen Content Provider erfolgen. Dieser nimmt die Speicherung in den applikationsinternen Dateien des Gerätes vor. Diese internen Dateien können nur durch die Applikation selbst gelesen und manipuliert werden (sichergestellt durch das Betriebssystem). Nicht einmal der Benutzer selbst hat Zugriff auf diese Dateien [33].

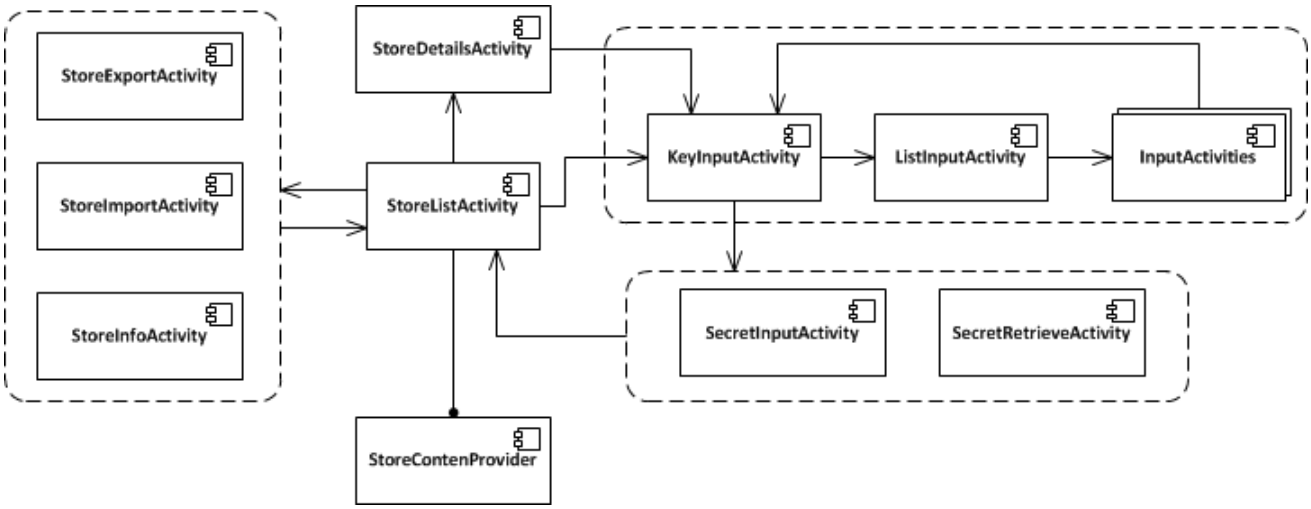


Abbildung 4.9.: Android KryptonIT Komponenten und Abhängigkeiten.

Kommunikation zwischen Activities

Die Kommunikation zwischen den verschiedenen Arten von Android Komponenten wird mit sogenannten Intents realisiert. Intents sind im Grunde Nachrichten, welche vom Betriebsystem verwaltet werden [34].

Um eine andere Activity aufzurufen, werden von Android verschiedene Möglichkeiten zur Verfügung gestellt: explizites oder implizites Starten. Bei der impliziten Variante, wird erst zur Laufzeit entschieden, welche Komponente gestartet wird (Late Binding). Für unsere Zwecke verwenden wir aus Gründen der Sicherheit das explizite Starten, wo wir auch den Klassen-Namen der Ziel-Activity benötigen. Im Gegensatz zum impliziten Starten einer Activity, bei welcher sich mehrere Activities auf einen Intent registrieren können und der Benutzer entscheiden muss welche er verwenden möchte.

Die benötigten Parameter, in unserem Fall die Zwischenresultate der KryptonIT Methode, werden mit Intent-Extras zwischen den Activities weitergegeben. Die Zwischenresultate sind als Modell-Klassen unserer Fassade abgebildet und können, weil diese serialisierbar sind, ohne Weiteres einem Intent hinzugefügt werden.

Sicherheit der Systemkomponenten

Jede Applikation erhält in Android eine eigene Linux User-ID. Der Linux Kernel des Android Systems stellt sicher, dass jede Applikation in einem eigenen Betriebssystem Prozess mit der entsprechenden User-ID voneinander getrennt laufen. Möchte eine Applikation Daten von einer Anderen verwenden, so muss dies explizit implementiert werden (z.B. über einen Content Provider).

Android schützt die Ressourcen und Operationen auf der Basis von Permissions. Der Benutzer kriegt bei der Installation einer neuen Applikation diese Permissions angezeigt und muss bestätigen, dass er die von der Applikation geforderten Permissions erlaubt.

Zusätzlich zu den Permissions, welche von Android definiert werden, kann eine Applikation eigene Permissions definieren. Mit diesen kann jede von der Applikation definierte Android Komponente geschützt werden, um die Nutzung der Komponenten zu kontrollieren.

Diesen Mechanismus wollen wir uns zu Nutze machen, indem wir alle Services und den Content Provider mit einer eigenen Permission markieren. So ist sichergestellt, dass ein Benutzer von KryptonIT den Zugriff auf seine Stores durch eine andere Applikation explizit bestätigen muss [35].

4.9.5. Persistierung von Daten

Für Android gibt es verschiedene Möglichkeiten Daten persistent zu Speichern. Jede Applikation besitzt einen privaten Speicher, auf welchen nur die betreffende Applikation selbst Zugriff hat. Weiter gibt es die Möglichkeit, Daten in den öffentlichen Bereich des Systems zu Speichern, welche von allen Applikationen und auch dem Benutzer selbst gelesen und modifiziert werden können [36].

Die Daten können theoretisch in einem beliebigen Format verwaltet werden. Android bietet die Möglichkeit an, eine SQLite Datenbank zu verwenden. Für unsere Zwecke legen wir die Daten der Secret Stores in den privaten Speicher ab und gewähren anderen Applikationen den Zugriff über einen Content Provider. So können wir selbst kontrollieren, welche Operationen durchführbar sind. Also auch, welche Felder aktualisiert werden können und z.B. ob das Aktualisieren/Löschen von mehreren Stores gleichzeitig möglich ist.

SQLite Datenbanken

SQLite ist ein in-process Datenbanksystem, welches sich für die Verwendung auf einem Gerät mit limitierten Speicher besonders gut eignet. Es besitzt einen sehr kleinen Footprint und bietet trotzdem die wichtigsten Funktionen eines DBMS [37].

Die Nutzung von SQLite, gegenüber gewöhnlichen Dateien, erspart uns die Handhabung von Dateisystem-Operationen und das Parsen von Inhalten.

Datentypen werden von SQLite etwas anders gehandhabt, als wir es uns von anderen DBMS gewohnt sind. Es werden grundsätzlich nur vier verschiedene Datentypen (werden in SQLite Speicher-Klassen gennant) angeboten. Auf dem Dateisystem, werden die Daten möglichst effizient abgespeichert, sobald die Daten aber abgerufen werden, werden diese im allgemeinsten Typ verarbeitet. Die verschiedenen von SQLite unterstützten Speicher-Klassen sind im Anhang C.3.2 aufgeführt.

4.9.6. Prozesse und Threads

Wie schon erwähnt, basiert Android auf Linux. Dadurch sind die Konzepte eines modernen Betriebssystems in Bezug auf parallele Verarbeitung und die Separation von Speicher übertragbar. Jede Applikation wird durch mindestens einen Prozess repräsentiert. Innerhalb eines Prozesses wird die Nebenläufigkeit (Parallelität) von Aufgaben durch Threads abgebildet. Wird eine neue Applikation in Android gestartet, läuft diese standardmässig in einem einzigen Prozess. Es ist zwar möglich die verschiedenen Komponenten, welche die Applikation beinhaltet, in separaten Prozessen laufen zu lassen. Im Moment sehen wir aber keinen Grund von dieser Möglichkeit Gebrauch zu machen.

Eine Android Applikation wird standardmässig in einem Thread gestartet, dem sogenannten „main“ Thread. In diesem werden alle UI Ereignisse verarbeitet und deshalb darf dieser Thread nicht blockiert werden (zum Beispiel durch die benötigten Rechenschritte beim Erstellen eines Secret Stores). Da nicht automatisch für jede Komponente ein separater Thread gestartet wird, muss sichergestellt werden, dass alle rechenintensiven Aufgaben in einem separaten Thread gestartet werden. Dauert eine Berechnung länger als 5 Sekunden, präsentiert das Android System dem Benutzer den berühmten „application not responding“ Dialog [38]. Da nicht garantiert ist, dass die Berechnungen in der geforderten Zeit durchgeführt werden können, wird vorgesehen, alle Berechnungen in den Service Komponenten in einem eigenen Thread durchzuführen. Dazu können wir wie gewohnt mit Java Threads arbeiten oder das durch Android definierte Konzept von „AsyncTasks“ verwenden [39].

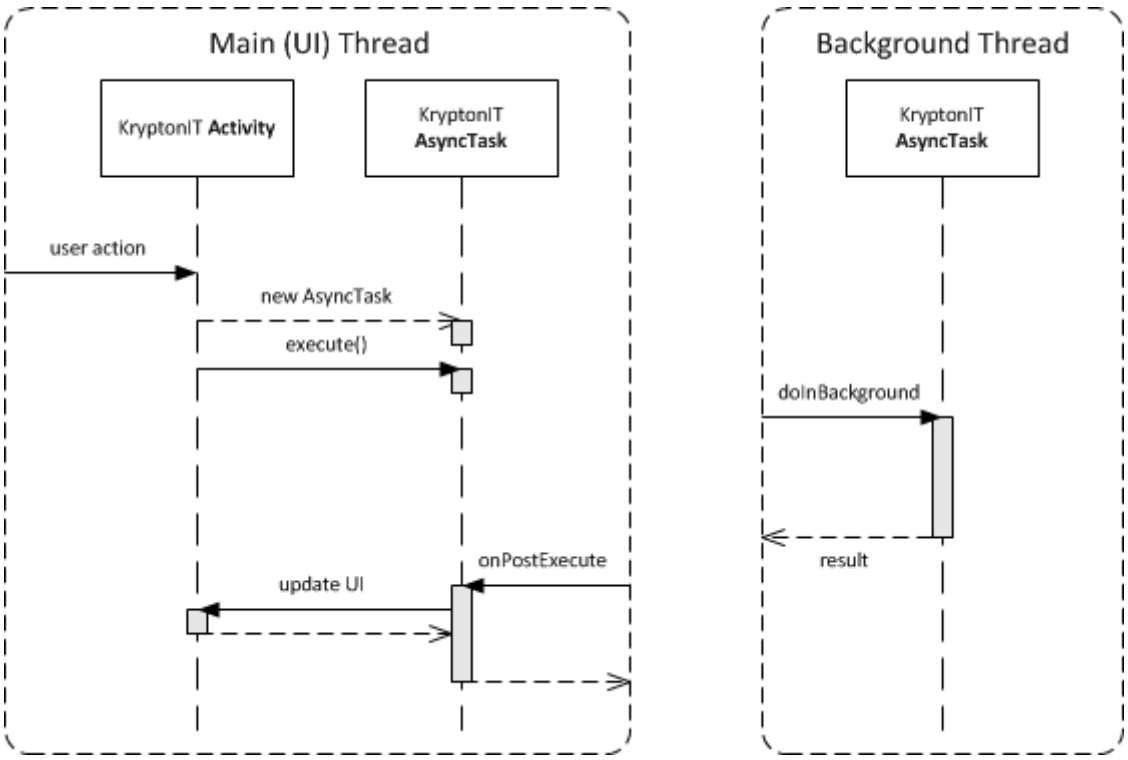


Abbildung 4.10.: Android KryptonIT Main und Worker Thread.

4.9.7. Tasks und Back Stack

Die sogenannten Tasks werden vom Android Bedienkonzept vorgegeben. Ein Task ist als Sammlung von Activities zu verstehen, welche vom Benutzer für eine bestimmte Aufgabe verwendet werden. Jeder Task besitzt einen „Back Stack“. Dies ist ein Stapel von Activities, welche in dem betreffenden Task aufgerufen wurden und in welcher der Benutzer mit der Zurück-Schaltfläche (jedes Android Gerät besitzt so einen Knopf, typischerweise unterhalb des Bildschirms angebracht) zurück Navigieren kann.

Entwickler einer Applikation können das Verhalten des Back Stack's kontrollieren. Das gewünschte Verhalten wird dem Intent, welcher für das Starten der nächsten Activity verwendet wird, gesetzt. So ist es z.B. möglich alle Activities vom Stapel zu löschen, wenn ein Task abgeschlossen ist [40].

4.9.8. Speichern des Activity Zustand

Als Entwickler einer Applikation muss der Zustand einer Activity zwischengespeichert werden, da das Android Betriebssystem unter Umständen die Activity zerstört. Zum Beispiel wenn die Activity nicht im Vordergrund ist und das Betriebssystem Arbeitsspeicher benötigt. Bei einer Konfigurations-Änderung (z.B. der Wechsel der Bildschirm-Orientierung von Hochformat zu Querformat) wird die Activity im Vordergrund zerstört und neu erstellt. Für einfache Werte, wie zum Beispiel der Text eines Eingabe-Feldes, wird das vom Betriebsystem automatisch gehandhabt. Für einen komplexen Zustand, wie zum Beispiel eine Liste von Schlüsseln, muss der Entwickler selbst sorgen. Android bietet da die Möglichkeit, die Methode `onSaveInstanceState` der Klasse `Activity` zu überschreiben. Diese wird vom System vor der Zerstörung der Activity aufgerufen und dann beim erneuten Erstellen wieder mitgegeben. So kann der vorherige Zustand wiederhergestellt werden, damit die Eingaben des Benutzers nicht verloren gehen [41], [42].

4.9.9. Applikations Ressourcen

Das Android SDK bietet an, die Ressourcen der Applikation (Texte, Bilder, UI Layout, etc.) vom Java Quellcode zu trennen. Dafür steht in jedem Android Projekt ein Verzeichnis `res` zur Verfügung. Bei den Ressourcen unterscheidet man zwischen default und alternativen. Alternative Ressourcen werden für die Unterstützung verschiedener Sprachen, aber auch für die Unterstützung verschiedener Bildschirmgrössen, Keyboard, Touchscreen Möglichkeiten, etc. eingesetzt. [43]

In einem ersten Schritt werden wir ausschliesslich default Ressourcen zur Verfügung zu stellen, da der Aufwand zum Erstellen alternativer Ressourcen den Rahmen dieses Projekts sprengen würde.

Lokalisierung und Internalisierung

Texte, welche auf der Benutzeroberfläche angezeigt werden, können wir in englischer sowie deutscher Sprache zur Verfügung stellen. Die Zeit, sowie das Knowhow für die Übersetzung in andere Sprachen steht uns aber nicht zur Verfügung [44].

- `res/values/strings.xml` : Default Ressourcen in englischer Sprache.
- `res/values-de/strings.xml` : Alternative Ressource in deutscher Sprache.

4.9.10. Unit und Blackbox Tests

Vom Android SDK wird eine auf JUnit basierende Testinfrastruktur zur Verfügung gestellt. Für uns sind vor allem die Testklassen `ActivityInstrumentationTestCase2` und `ProviderTestCase2` interessant. Welche es uns erlauben eine Activity- oder ContentProvider-Komponente als Einheit zu testen.

Für die Paketierung und Ausführung der Tests bieten das SDK und das Eclipse Plugin ADT Hilfsmittel an. Die Testklassen werden in einem separaten Eclipse Projekt erstellt. Aus diesem Projekt resultiert eine spezielle Test-Applikation, welche über die Klasse `InstrumentationTestRunner` die Komponenten in der zu testende Applikation aufruft/fernsteuert [45].

Das quelloffene Projekt Robotium hingegen stellt ein Framework für die Umsetzung und Durchführung von Blackbox Tests zur Verfügung. Dieses erlaubt, auf der Basis der Android Testinfrastruktur, Benutzeraktionen (z.B. einen Klick auf einer Schaltfläche, oder die Eingabe von Text) zu automatisieren. Dabei ist kein Wissen über den Aufbau der Applikation notwendig, da für die Identifizierung der Steuerelemente ausschliesslich ein Text (z.B. die Beschriftung einer Schaltfläche) oder die Reihenfolge (z.B. zweites Element in einer Liste) verwendet wird [46].

4.9.11. Android Software-Keyboards

Leider ist die Eingabe von Zahlen zwischen verschiedenen Android Versionen nicht konsistent. In Version 2.2 und 2.3.1 wird das Keyboard wie in Abbildung 4.11a angezeigt. Obwohl nur Zahlen eingegeben werden können, werden auf dem Keyboard Sonderzeichen angezeigt. Erst ab Version 2.3.3. wird die Eingabe mit einem angepassten Softkeyboard, siehe Abbildung 4.11a, umgesetzt.

Das Android Betriebssystem ermöglicht es, ein neues Soft-Keyboard zu installieren. Der Benutzer kann das System so konfigurieren, dass dieses systemweit für die Eingabe verwendet wird. Bekannte Vertreter dieser Keyboard Erweiterungen sind SwiftKey[1] und Swype[2]. Wir möchten die Eingabe von Zahlen aber nur für bestimmte Eingaben innerhalb unserer Applikation benutzerfreundlicher machen. Als systemweiten Ersatz wäre dieses Nur-Zahlen Keyboard von geringem Nutzen. Android erlaubt es, für spezifische Eingabe-Felder ein eigenes Keyboard zur Verfügung zu stellen [47]. Ein Keyboard kann als XML-Datei über die Klasse `Keyboard` [48] definiert und anschliessend mit `KeyboardView` angezeigt werden.

Android bietet aus unserer Sicht keine optimale Lösung für die Eingabe. Es wäre möglich, ein eigenes Keyboard zur Verfügung zu stellen. Der benötigte Aufwand für diese Erweiterung kann aber nicht durch einen enormen Gewinn an Benutzerfreundlichkeit gerechtfertigt werden. Deshalb wird im Moment auf eine solche Umsetzung verzichtet und mit den Keyboards von Android gearbeitet.

(a) Android Version 2.2 und 2.3.1 (b) Android Version 2.3.3

Abbildung 4.11.: Android Keyboards

[1] SwiftKey, `http://www.swiftkey.net/`
[2] Swype, `http://www.swype.com/`

4.9.12. QR Code

Ein QR Code (QR steht für Quick Response) ist ein zweidimensionaler Code. Er besteht aus einer Matrix aus weissen und schwarzen Punkten und enthält in drei der vier Ecken eine spezielle Markierung um die Orientierung des Codes vorzugeben [49].

Für Android steht eine Implementation für QR Codes in Form einer Applikation bereit. Der „Barcode Reader“ von ZXing [50]. Dieser kann von beliebigen Applikationen verwendet werden, um QR Codes zu lesen und auch zu erstellen. Dies möchten wir für zwei Anwendungszwecke einsetzten. Einerseits für den Import, resp. Export von Stores und andererseits als Schlüssel-Eingabe.

4.9.13. Near Field Communication

Near Field Communication (NFC) bezeichnet einen kontaktlosen Übertragungsstandard für sehr kurze Strecken (ca. 4cm). Die Kommunikation findet zwischen sogenannten Tags und einem Endgerät statt. In unserem Fall ist das Gerät ein Android Smartphone. Die Android Plattform bietet seit neuerem (ab Version 2.3) Unterstützung für NFC. Leider gibt es noch wenig Geräte, welche mit einem NFC-Modul ausgeliefert werden. In der Schweiz sind einzig die sog. Google-Phones, welche als Referenz-Design für die jeweilige Version auf den Markt gelangen, mit einem entsprechenden Modul ausgestattet [51].

Die Android Plattform unterstützt verschiedene *Tag* Technologien und Klassen. Die BFH-Card, welche als Studentenausweis und als Zugangs-Karte innerhalb der BFH eingesetzt wird, ist auch ein NFC Tag. Das Tag wird sogar von Android unterstützt. Zusammen mit einem von der Schule zur Verfügung gestellten Android Gerät, dem Google Nexus S (das Gerät enthält ein NFC Modul), möchten wir die BFH-Card auslesen und als Schlüssel-Eingabe verwenden.

Bei der Analyse haben wir festgestellt, dass die BFH Card vom Typ *NFC-V* (definiert in ISO 15693) ist. Dieser Typ wird von Android nur rudimentär unterstützt. Die Kommunikation mit dem Tag muss auf Byte-Ebene implementiert werden und erfordert damit das Studium des entsprechenden Standards. Einzig die NDEF Formatierten Tags sind voll implementiert und deren Inhalt kann einfach über das API ausgelesen werden [52]. Wir haben uns entschieden, das Thema NFC für Eingaben nicht weiter zu verfolgen, da das Thema zu komplex ist.

5. Design

5.1. Web Applikation

Das Design der Web Applikation besteht aus vier Seiten. Die erste Seite ist die Einleitungsseite, auf welcher erklärt wird, worum es geht. Auf der zweiten Seite kann ein Store erstellt und gespeichert werden. Auf der dritten Seite kann ein Store geladen und abgefragt werden. Auf der vierten Seite kann ein Store importiert, exportiert oder gelöscht werden.

Die Seiten „Store erstellen“ und „Store abfragen“ werden jeweils in drei nebeneinander liegende Bereiche aufgeteilt. Im linken Bereich der Seite befindet sich die Grundapplikation, worin der Benutzer zwischen den einzelnen Schritten eines Vorgangs wechseln und die dafür benötigten Werte und Aktionen eingeben und ausführen kann. In der Mitte wird eine Grafik angezeigt, welche ein Koordinatensystem und zwei weitere Werte-Achsen darstellt, worauf die einzelnen Schritte grafisch aufgezeigt werden. Im rechten Bereich der Seite ist der Informationsbereich, welcher bei jedem Erstellungs- oder Abfrageschritt die Resultate der durchgeführten Berechnungen anzeigt.

5.1.1. Einleitungsseite

Die Einleitungsseite ist die Einstiegsseite der Applikation. Der Benutzer wird kurz begrüsst und findet eine Einleitung, welche Beschreibt, was die Idee der Applikation ist und was der Benutzer mit der Applikation machen kann.

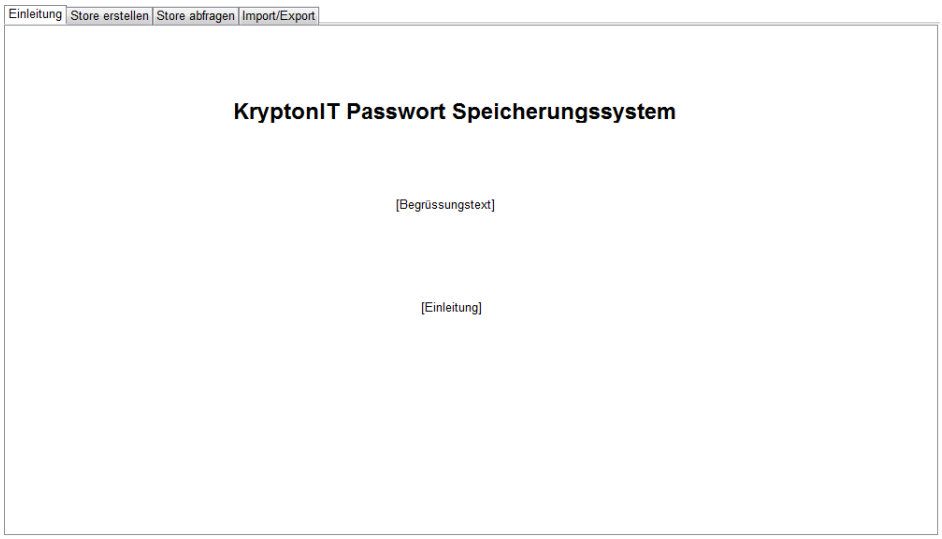


Abbildung 5.1.: Einleitungsseite der Applikation

5.1.2. Store erstellen Seite

Auf der „Store erstellen“ Seite (siehe Abbildung 5.1) werden die einzelnen Schritte aufgezeigt, welche beim Erstellen eines Stores durchlaufen werden.

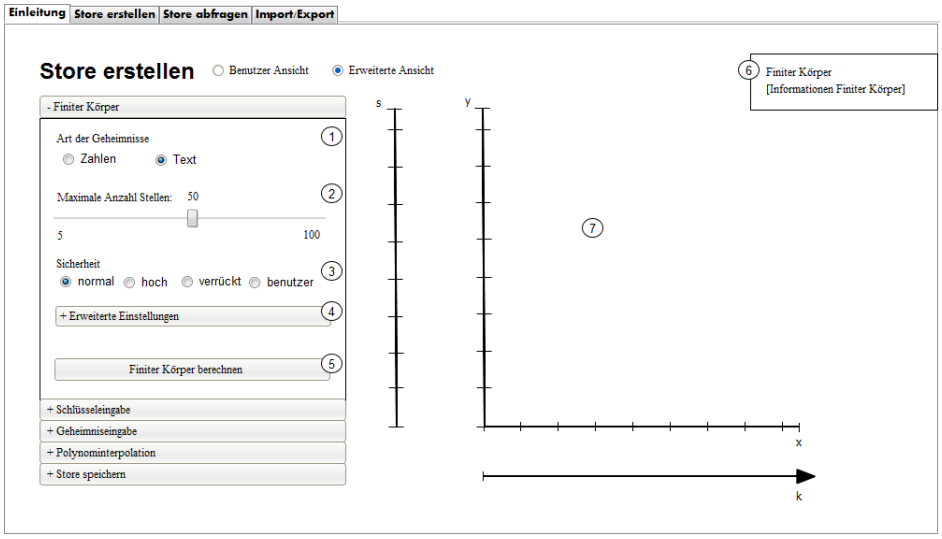


Abbildung 5.2.: Schritt 1: Definieren des finiten Körpers

Als erster Schritt (siehe Abbildung 5.2) muss der finite Körper definiert werden. Dafür muss die Art (1) und die maximale Länge der Geheimnisse (2) festgelegt und die gewünschte Sicherheit (3) eingestellt werden. In den „Erweiterten Einstellungen“ (4) kann die Sicherheit individuell definiert werden. Sind alle benötigten Eingaben gemacht, kann der finite Körper durch Klicken auf den Knopf „Finiter Körper berechnen“ (5) berechnet werden. Die Resultate der durchgeführten Berechnungen werden im Informationsbereich (6) angezeigt und im Koordinatensystem (7) wird die Beschriftung der Geheimnis- (s-), y- und x-Achse ergänzt.

Erweiterte Einstellungen

Abbildung 5.3.: Schritt 1: Erweiterte Einstellungen

Die erweiterten Einstellungen (siehe Abbildung 5.3) müssen nicht zwingend verändert werden, da Sie auf vordefinierte Werte gesetzt wurden, indem die Auswahl der Sicherheit bei den Grundeinstellungen betätigt wurde. Möchte man aber eigene Einstellungen verwenden, können diese in den erweiterten Einstellungen durch Anpassen der beiden Sicherheitsparameter und der Hash Methode individuell eingestellt werden.

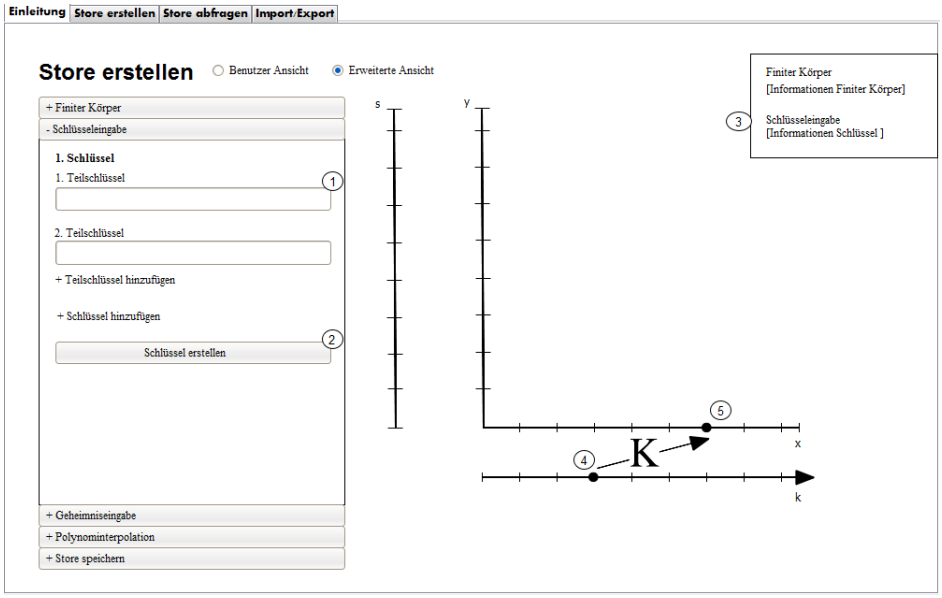


Abbildung 5.4.: Schritt 2: Schlüsseleingabe

Als zweiter Schritt (siehe Abbildung 5.4) müssen die Schlüssel eingegeben werden (1), mit welchen man später die gespeicherten Geheimnisse abfragen kann. Ein Schlüssel kann aus einem oder mehreren Teilschlüsseln erstellt werden. Hat man für jedes zu speichernde Geheimnis mindestens einen Teilschlüssel definiert, kann der Knopf „Schlüssel erstellen“ (2) geklickt werden. Dadurch werden die Teilschlüssel zu einem Schlüssel kombiniert. Die Resultate der durchgeführten Berechnungen werden im Informationsbereich (3) angezeigt und beim Koordinatensystem wird die Beschriftung der Schlüssel-Achse (k) ergänzt und die berechneten Schlüssel-Werte werden als Punkte auf der Schlüssel-Achse eingetragen (4). Weiter werden die Schlüssel auf die x-Achse in den Bereich des finiten Körpers (maximale Werte des Koordinatensystems) transformiert. Die Berechneten x-Werte der Schlüssel werden im Informationsbereich angezeigt (3). Im Koordinatensystem werden die transformierten Schlüssel auf der x-Achse als Punkte eingetragen (5).

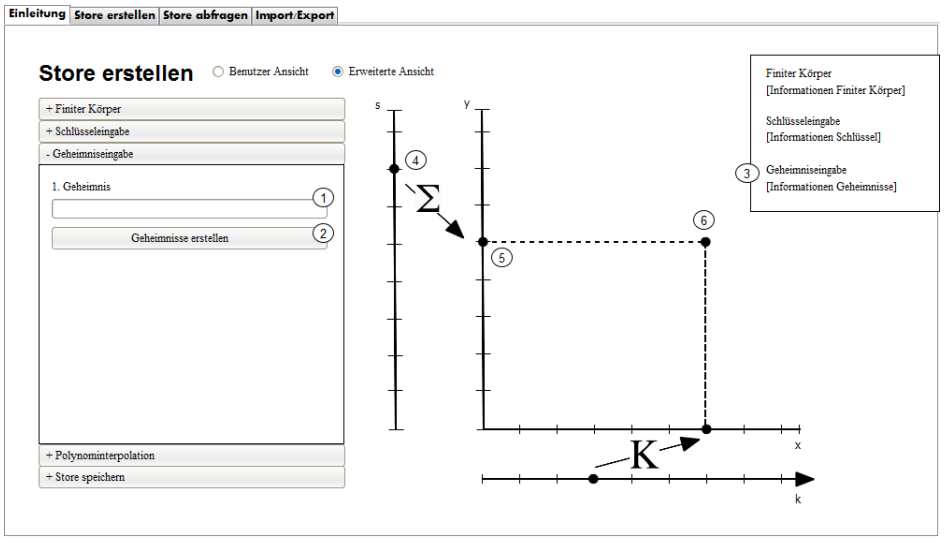


Abbildung 5.5.: Schritt 3: Geheimniseingabe

Als dritter Schritt (siehe Abbildung 5.5) müssen nun die Geheimnisse für die definierten Schlüssel eingegeben werden (1). Durch Klicken auf den Knopf „Geheimnisse erstellen“ (2), werden die Geheimnis-Werte

berechnet und im Informationsbereich angezeigt (3). Im Koordinatensystem werden die Geheimnisse als Punkte auf der Geheimnis-Achse eingetragen (4). Als Nächstes werden die Geheimnisse auf die y-Achse in den Bereich des finiten Körpers transformiert. Die berechneten y-Werte werden im Informationsbereich angezeigt (3). Im Koordinatensystem werden die transformierten Geheimnisse auf der y-Achse als Punkte eingetragen (5). Weiter werden die Punkte, welche durch die gesetzten x- und y-Werte der einzelnen Schlüssel-Geheimnis Paare entstanden sind, im Koordinatensystem als Punkte eingezeichnet (6).

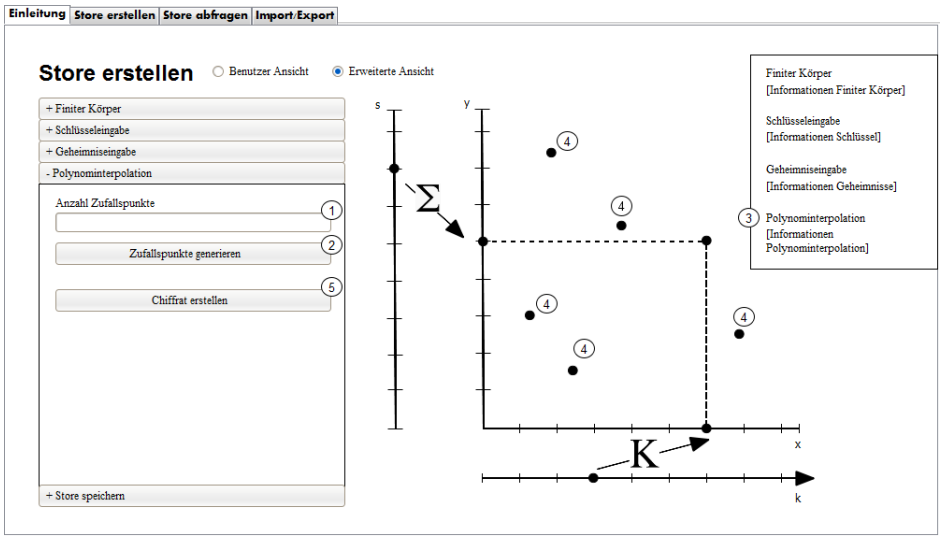


Abbildung 5.6.: Schritt 4: Polynominterpolation

Als vierter Schritt (siehe Abbildung 5.6) wird der Secret Store erstellt. Dafür muss noch definiert werden, wie viele zusätzliche Zufallspunkte für die Generierung des Polynoms erstellt werden sollen (1). Durch Klicken auf den Knopf „Zufallspunkte generieren“ (2), werden die gewünschte Anzahl zusätzlicher Punkte erzeugt. Im Informationsbereich werden die generierten Punkte angezeigt (3) und im Koordinatensystem werden alle generierten Punkte eingetragen (4). Als Letztes muss auf den Knopf „Chiffrat erstellen“ geklickt werden (5). Das Polynom, welches den Secret Store darstellt, wird generiert und im Informationsbereich angezeigt (3).

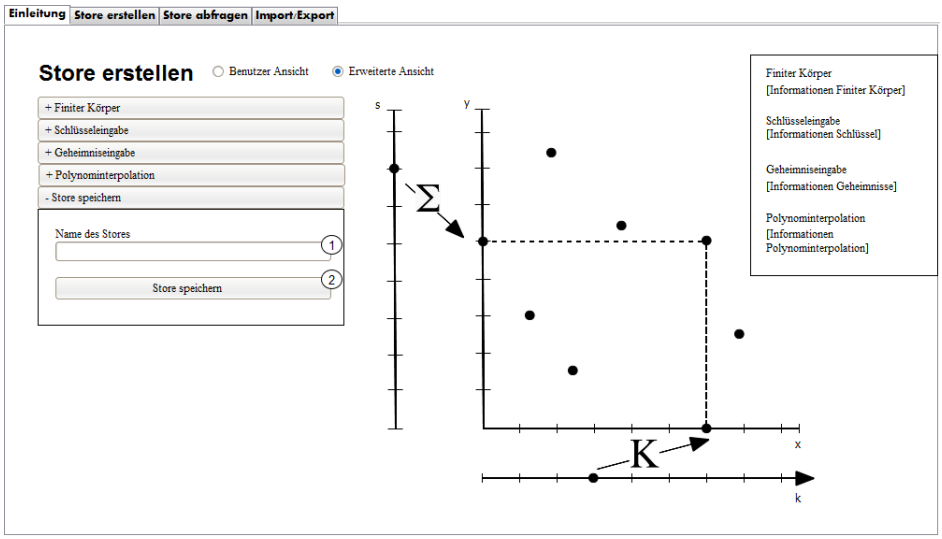


Abbildung 5.7.: Schritt 5: Store speichern

Als fünfter Schritt (siehe Abbildung 5.7) kann der Store gespeichert werden. Zuerst muss im Eingabefeld ein Name für den zu speichernden Store definiert werden (1) und dann noch der Knopf „Store speichern“ geklickt werden (2). Der Benutzer erhält eine Information, dass der Store erfolgreich gespeichert wurde.

5.1.3. Store abfragen Seite

Auf der „Store abfragen“ Seite werden die einzelnen Schritte aufgezeigt, welche beim Abfragen eines Geheimnisses aus einem Store durchlaufen werden.

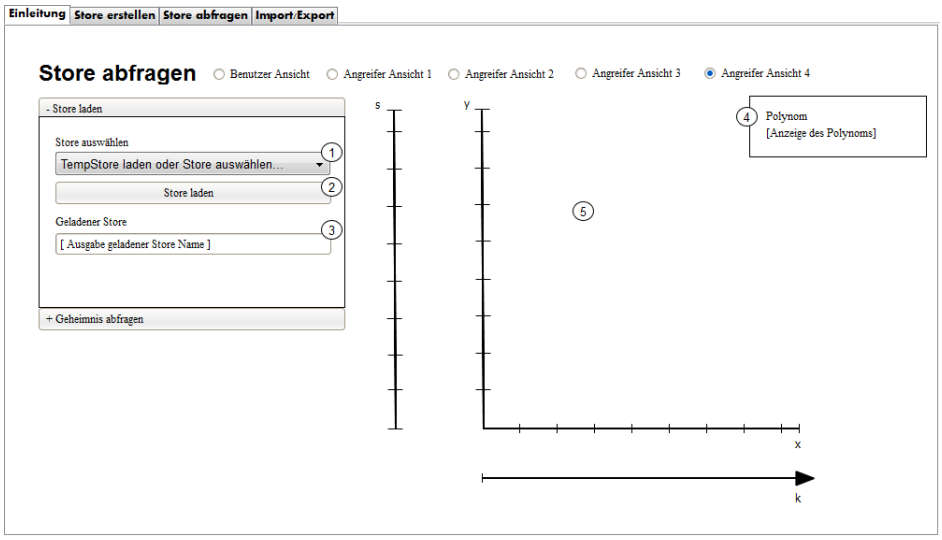


Abbildung 5.8.: Schritt 1: Store laden

Als erster Schritt (siehe Abbildung 5.8) muss der Store, welcher abgefragt werden soll, selektioniert (1) und geladen (2) werden. Hat man einen Store geladen, wird dessen Name im Ausgabefeld angezeigt (3). Die Informationen zum geladenen Store werden im Informationsbereich (4) angezeigt und im Koordinatensystem (5) wird die Beschriftung der Geheimnis- (s-), y- und x-Achse ergänzt.

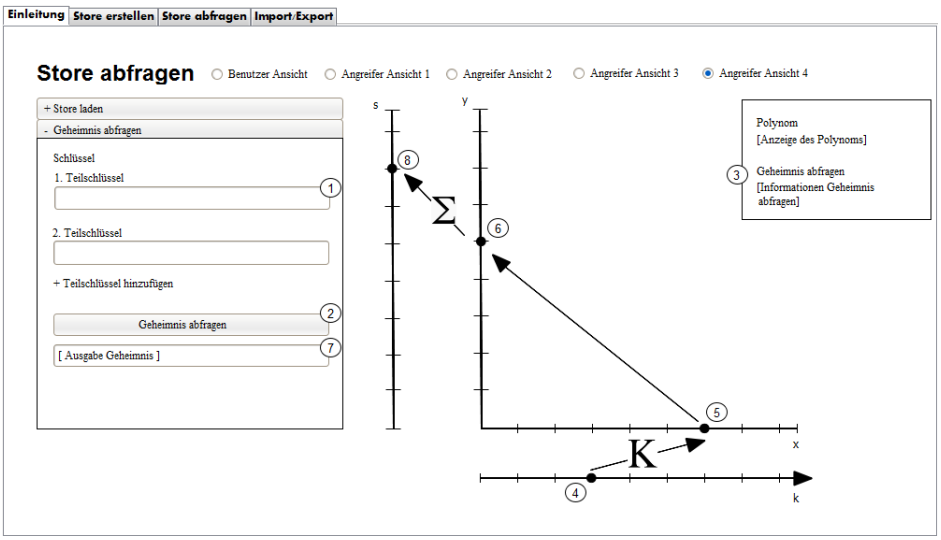


Abbildung 5.9.: Schritt 2: Store abfragen

Als zweiter Schritt (siehe Abbildung 5.9) muss der Schlüssel eingegeben werden (1), mit welchem man das zugehörige Geheimnis abfragen will. Hat man alle Teilschlüssel eingeben, muss auf den Knopf „Geheimnis abfragen“ (2) geklickt werden. Hierbei werden die Teilschlüssel miteinander kombiniert. Die Resultate der durchgeführten Berechnungen werden im Informationsbereich (3) angezeigt und im Koordinatensystem wird die Beschriftung der Schlüssel-Achse (k-Achse) ergänzt und der berechnete Schlüssel-Wert wird als Punkt auf der Schlüssel-Achse eingetragen (4). Weiter wird der Schlüssel auf die x-Achse in den Bereich des finiten Körpers (maximale Werte des Koordinatensystems) transformiert. Der berechnete x-Wert des Schlüssels wird im Informationsbereich angezeigt (3). Im Koordinatensystem wird der transformierte Schlüssel als Punkt auf der x-Achse eingetragen (5). Als Nächstes wird der x-Wert in das Polynom eingesetzt und der berechnete y-Wert wird im Informationsbereich (3) angezeigt und im Koordinatensystem auf der y-Achse als Punkt eingetragen (6). Als Letztes wird der erhaltene y-Wert des Geheimnisses noch in das effektive Geheimnis zurück transformiert. Das gesuchte Geheimnis wird im Ausgabefeld (7) unter dem Knopf angezeigt und auf der Geheimnis-Achse eingetragen (8).

5.1.4. Import/Export Seite

Auf der Import/Export Seite kann ein Store importiert, exportiert oder gelöscht werden.

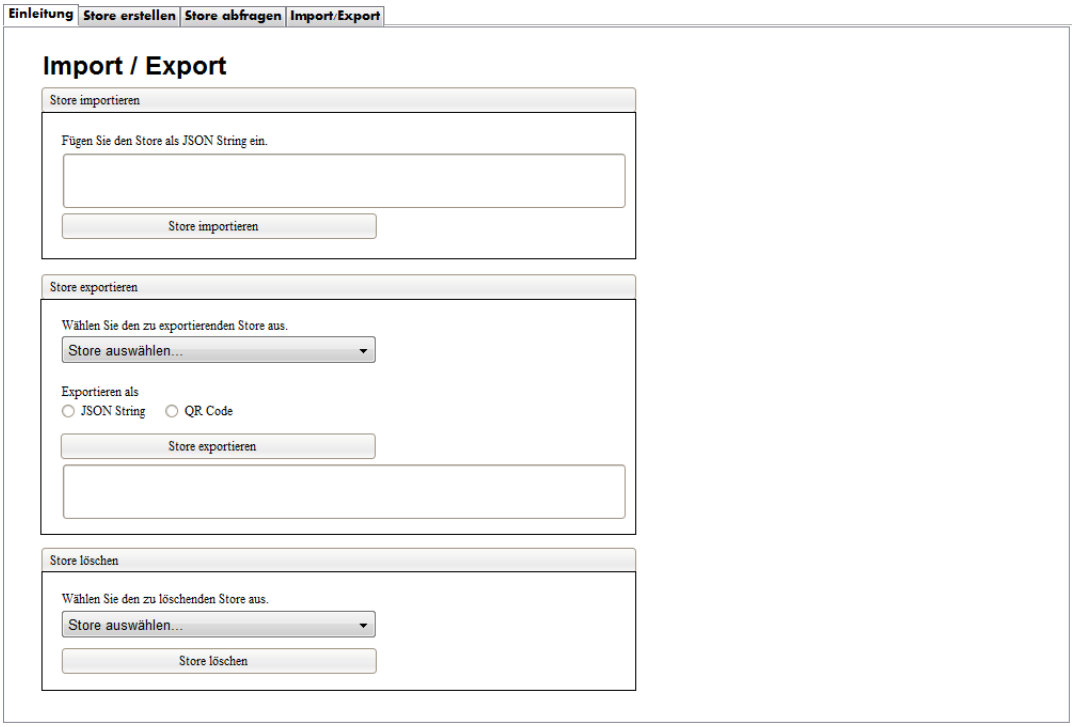


Abbildung 5.10.: Import/Export Seite

Store importieren
Um einen Store zu importieren, muss dieser als String im JSON-Format in das mehrzeilige Textfeld eingegeben und dann auf den Knopf „Store importieren“ geklickt werden. Es wird eine Meldung angezeigt, dass der Store erfolgreich importiert wurde.

Store exportieren

Um einen Store zu exportieren, muss dieser zuerst ausgewählt werden. Weiter muss bestimmt werden, ob des Store als QR-Code oder als Text (JSON-String) exportiert werden soll. Durch Klicken auf den Knopf „Store exportieren“, wird der Store im gewünschten Export-Format exportiert und angezeigt.

Store löschen

Um einen Store zu löschen, muss dieser zuerst ausgewählt werden. Durch Klicken auf den Knopf „Store löschen“, wird der selektionierte Store gelöscht.

5.2. Mobile Applikation

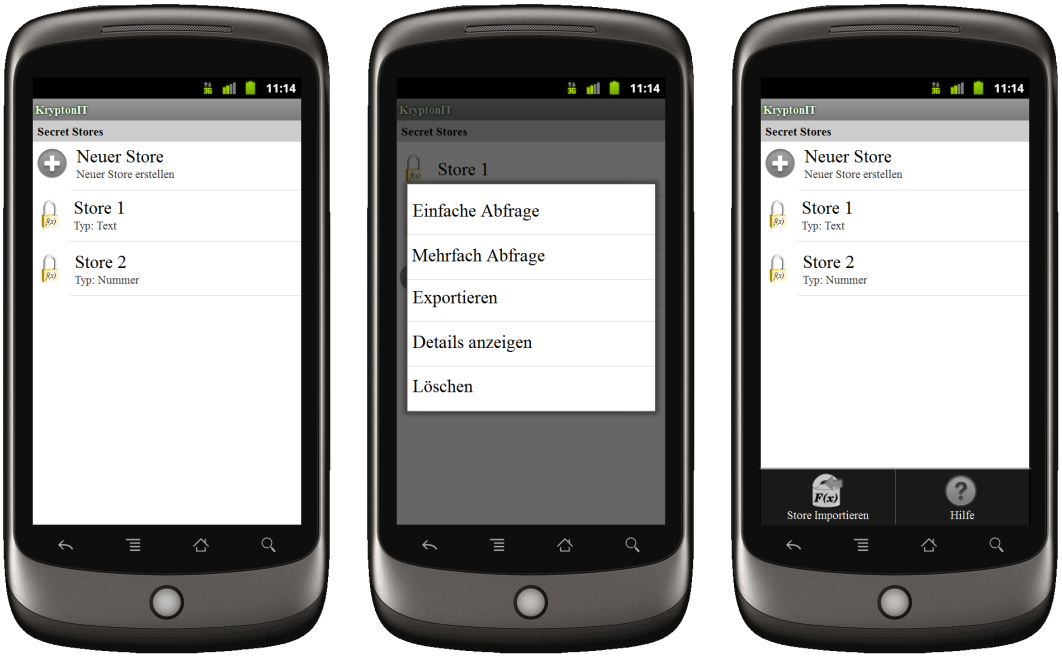
5.2.1. Benutzeroberfläche und Usability Design

Wir wollen für den Benutzer einen möglichst einfachen Zugang zu der KryptonIT Methode anbieten. Gleichzeitig möchten wir möglichst viele von der Android Plattform zur Verfügung gestellten UI Komponenten verwenden, um so nahe wie möglich an das Android Look-and-feel zu kommen.

Wenn die Applikation gestartet wird, wird dem Benutzer als Erstes der zentrale Bildschirm 5.11a angezeigt. Dieser erlaubt es, alle Funktionen rund um die Secret Stores auszuwählen. Die vorhandenen Stores werden zusammen mit der Option „Neuer Store“ in einer Liste dargestellt, wobei „Neuer Store“ immer an erster Stelle der Liste steht. Wählt der Benutzer die Option „Neuer Store“, wird er durch den Erstell-Prozess geführt (siehe auch Abbildung 5.18). Wird ein bestehender Store angewählt, so wird die „Einfache Abfrage“ durchgeführt (siehe Abbildung 5.20).

Durch einen sogenannten long-press (längeres Anwählen eines Elements) erscheint hingegen das Kontextmenü 5.11b, welches weitere Store-spezifische Optionen anbietet. Der erste Menü-Punkt „Einfache Abfrage“ ist identisch mit der normalen Anwahl (kurzes Anwählen eines Stores). Der zweite Punkt „Mehrfach Abfrage“ erlaubt es, mehrere Geheimnisse in einem Durchlauf abzufragen (siehe Abbildung 5.19). Mit dem Eintrag „Exportieren“ wird die Export-Funktion angezeigt (siehe 5.2.1). Der Menü-Punkt „Details anzeigen“ zeigt die gespeicherten Informationen über den Store an. Am Ende ist noch der Menü-Punkt „Löschen", welcher den gewählten Store vom Applikations-Speicher löscht.

Bei der Auswahl der Android Menü-Taste erscheint das Optionsmenü, welches in Abbildung 5.11c dargestellt wird. Dieses bietet an, einen bestehenden Store zu importieren, sowie die integrierte Hilfe der Applikation aufzurufen (siehe 5.2.1).



(a) Auflistung der vorhandenen Stores (b) Kontext-Menü für einen Store (c) Menü der Hauptmaske

Abbildung 5.11.: Android UI Design 1

Store Parameter

Bei der Erstellung eines Stores, müssen zuerst dessen Parameter festgelegt werden. Die beiden Masken 5.12a und 5.12b sind über ein Register-Steuerelement verbunden. Standardmässig ist die Maske 5.12a ausgewählt, welche die wichtigsten Parameter vereinfachend zusammenfasst. Neben dem Namen, der Art der Geheimnisse und der Anzahl Stellen, können die Sicherheitsparameter auf dem zweiten Register über die drei voreingestellten Garnituren „Normal'", „Hoch" und „Verrückt" gewählt werden. Die Wahl beeinflusst direkt die Werte auf der Maske 5.12b. Wechselt der Benutzer das Register auf diese Maske, so kann er die Werte für die Parameter „Sicherheitsparameter 1", „Sicherheitsparameter 2", die Anzahl der Zufallspunkte und die verwendete Hash-Methode wählen. Dabei wird aber die Voreinstellung dieser Parameter auf der Maske 5.12a deselektiert. Mit der Schaltfläche „Weiter" gelangt der Benutzer auf die Maske 5.13a für das Hinzufügen von Schlüsseln.

(a) Einstellungen für einen neuen Store (b) Erweiterte Einstellungen für einen neuen Store

Abbildung 5.12.: Android UI Design 2

Schlüssel Eingabe

Die Eingabe von Schlüsseln wird in zwei Modi vorgesehen. Die Eingabe einer beliebigen Anzahl von Schlüssel und deren Teilschlüssel. Und die Eingabe eines einzigen Schlüssels mit beliebig vielen Teilschlüsseln. Im ersten Modus, wird auf der Maske 5.13a als erstes Element in der Liste immer die Option „Schlüssel hinzufügen“ platziert. Bei der Wahl dieser Option, gelangt der Benutzer auf die Maske 5.13c wo der Eingabe-Typ ausgewählt wird. Da jeder Schlüssel potentiell aus mehreren Teilschlüsseln besteht, werden die Teilschlüssel eingerückt unterhalb des jeweiligen Schlüssel dargestellt. Für das Hinzufügen von Teilschlüsseln ist jeweils unter jedem Schlüssel-Eintrag ein Eintrag „Teilschlüssel hinzufügen“ eingeblendet. Wählt der Benutzer einen solchen Eintrag aus, so gelangt er, genau wie bei der Option „Schlüssel hinzufügen“, auf die Maske 5.13c.

Der Bildschirm 5.13c listet alle verfügbaren Eingabe Möglichkeiten auf. Zur visuellen Unterstützung des Benutzers wird pro Eingabe Typ eine sprechende Ikone angezeigt. Bei der Wahl eines Eintrages wird die entsprechende Eingabe Maske angezeigt. Nach der Eingabe eines Schlüssels werden alle bereits eingegebenen Schlüssel und Teilschlüssel in der Maske 5.13a angezeigt. Wählt der Benutzer, weitere Schlüssel einzugeben, gelangt er wieder auf den Bildschirm 5.13c. Mit „Weiter“ wird die Eingabe von Schlüsseln abgeschlossen und der Benutzer gelangt zur Geheimnis Eingabe 5.14a.

In Bildschirm 5.13a ist die freie Eingabe einer beliebigen Zeichenkette dargestellt. Weitere Eingabe Masken werden in Kapitel 5.2.2 behandelt.

(a) Eingabe und Auflistung beliebig vieler Schlüssel

(b) Eingabe eines einzigen Schlüssels

(c) Auswahl der Eingabe-Typen

(d) Freie Eingabe eines Schlüssels

Abbildung 5.13.: Android UI Design 3

Geheimnis Ein- und Ausgabe

Die Eingabe von Geheimnissen (bei der Erstellung von Stores), wird mit der Benutzeroberfläche 5.14a abgedeckt. Für jeden eingegebenen Schlüssel wird ein Eintrag in der Liste erstellt, welcher mit einem Fragezeichen markiert ist. Sobald ein Wert eingegeben wurde, wird für den betreffenden Eintrag ein Gut-Zeichen dargestellt. Die Eingabe wird ausgelöst, indem ein Eintrag gewählt wird. Je nach Wahl der Geheimnis Art (siehe auch Abbildung 5.12a), wird dem Benutzer angeboten, Text (siehe 5.14c) oder eine Zahl (siehe 5.14b) einzugeben. Wenn alle Geheimnisse eingegeben wurden, kann der Store Erstell-Vorgang mit der Schaltfläche „Abschliessen“ abgeschlossen werden.

Die Ausgabe der Geheimnisse in Bildschirm 5.14d, wird sehr ähnlich wie die Eingabe aufgebaut. Bei jedem Geheimnis wird eine Status-neutrale Ikone dargestellt und der Wert als Text oder Zahl formatiert, je nach Geheimnis-Typ des aktuellen Stores.

(a) Geheimnis Eingabe (b) Eingabe eines Zahlen-Geheimnis (c) Eingabe eines Text-Geheimnis (d) Anzeige der abgefragten Geheimnisse

Abbildung 5.14.: Android UI Design 4

Hilfe

In jedem Bildschirm der Applikation wird ein Hilfe-Menü (welches über die Android Menü-Taste aufrufbar ist) zur Verfügung gestellt. Wählt der Beutzer dieses Menü aus, erscheint eine neue Ansicht, in welcher die Hilfe-Texte angezeigt werden.

Abbildung 5.15.: Android KryptonIT - Hilfe Menü

Export

Die Export Oberfläche 5.16 steht für jeden vorhandenen Store zur Verfügung und wird über das Menü aufgerufen (siehe Abbildung 5.11b). Die Maske zeigt das JSON Export-Format des gewählten Stores als Text an. Es werden drei Export Optionen angeboten: als QR Code, als Datei oder in die Zwischenablage von Android. Die beiden Erstgenannten werden über die entsprechende Schaltfläche gestartet. Das Speichern in die Zwischenablage wird beim Anwählen des angezeigten Export-Formats ausgelöst.

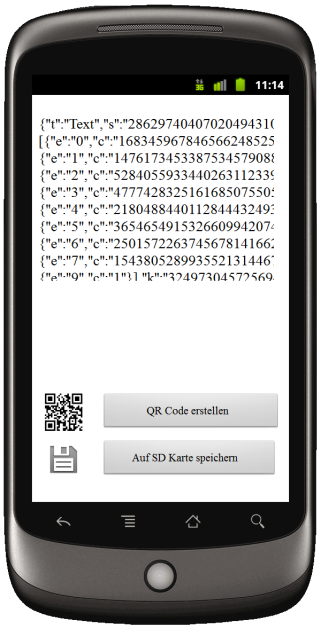


Abbildung 5.16.: Android KryptonIT - Export

Import

Die Import Funktionalität 5.17 wird über das Menü 5.11c aufgerufen. Und bietet als Gegenstück zur Export-Funktionalität drei Import Optionen an. Als erste Option wird das Einfügen vom JSON Export-Format in eine Text-Eingabe angeboten. Als Nächstes der Import von QR Code als Schaltfläche. Zu Unterst in der Maske wird eine Auswahl von vorhandenen Dateien aufgelistet und über die Schaltfläche „Von SD Karte importieren“ ausgelöst. Bei erfolgreichem Import wird der Benutzer jeweils wieder zurück auf die zentrale Maske geleitet.

Abbildung 5.17.: Android KryptonIT - Import

Screenflow

Hier ist der Ablauf und die Navigation zwischen den einzelnen Ansichten der Android Applikation, zusätzlich zu der textuellen Beschreibung weiter oben, visualisiert.

Die Erstellung eines Stores stellt den komplexesten Ablauf innerhalb der Applikation dar (siehe Abbildung 5.18). Nach der Eingabe der Store Parameter (zweiter Bildschirm von links), kann der Benutzer beliebig viele Schlüssel und Teilschlüssel hinzufügen, um dann für jeden eingegebenen Schlüssel ein Geheimnis einzugeben (die beiden Bildschirme ganz Rechts stellen die Geheimnis-Eingabe von Zahlen oder Text dar).

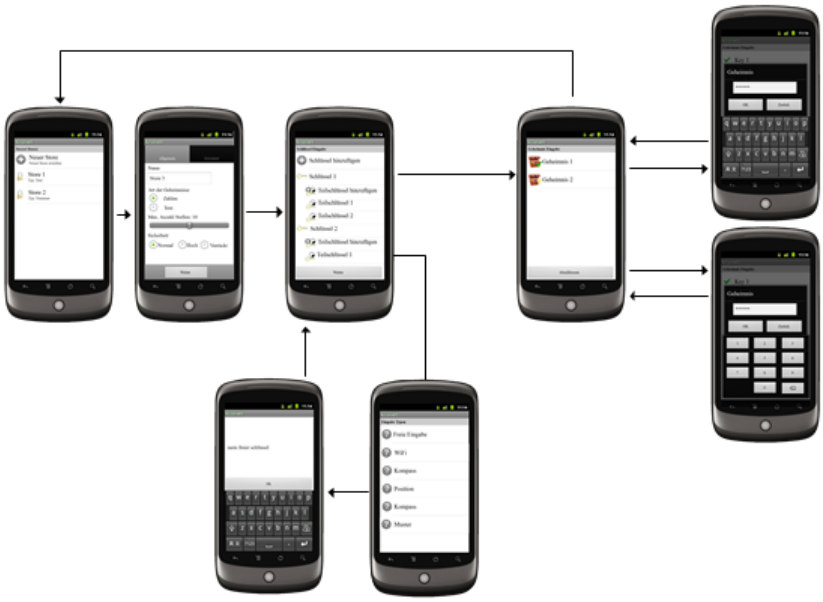


Abbildung 5.18.: Android KryptonIT - Screenflow für Store Erstellen

Die komplexe Abfrage eines Stores ist, bis auf die Eingabe von Geheimnissen und das Wegfallen der Store Parameter, identisch mit dem Ablauf bei der Erstellung.

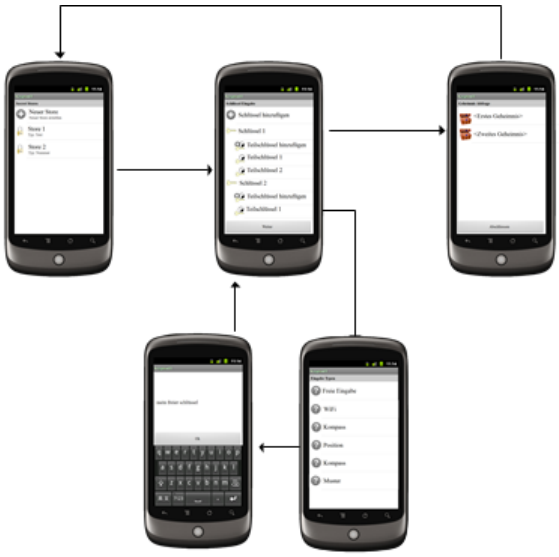


Abbildung 5.19.: Android KryptonIT - Screenflow für komplexe Store Abfrage

Bei der einfachen Abfrage wird der Benutzer direkt auf die Auswahl von Eingabe Typen geleitet, um den

Schlüssel für die Abfrage zu definieren. Es können auch keine weiteren Schlüssel hinzugefügt werden, sondern nur Teilschlüssel.

Abbildung 5.20.: Android KryptonIT - Screenflow für einfache Store Abfrage

5.2.2. Visualisierung der Schlüssel-Eingaben

Die folgenden Eingabe-Typen sind für die Eingabe als Schlüssel-Werte vorgesehen. Sie sind hier aufgeführt, weil sie besondere Ansprüche an die Visualisierung stellen.

WLAN Adressen

Ein WLAN AccessPoint wird vom Benutzer grundsätzlich über die SSID identifiziert und gewählt. Die SSID wird im AccessPoint konfiguriert und kann vom Administrator des WLAN Netzwerkes frei gewählt und nachträglich auch wieder geändert werden. Zusätzlich gibt es eine BSSID pro WLAN Netzwerk, welche der MAC-Adresse des Access Points entspricht. Die BSSID ist eine 48 Bit lange Adresse.

Diese 48 Bit Adresse können wir für die Anzeige von zwei verschiedenen 24 Bit Farben verwenden. Dabei werden die ersten 8 Bit der Addresse für Rot, die zweiten 8 Bit für Grün und die dritten für den blauen Farb-Kanal verwendet. Analog dazu wird die zweite Farbe mit den vierten bis sechsten 8 Bit der Adresse gebildet. Somit hat der Benutzer drei verschiedene Merkmale, um eine WLAN Adresse auszuwählen: die SSID, die BSSID und zwei aus der BSSID berechnete Farben.

Abbildung 5.21.: Visualisierung von WLAN Access Points

Himmelsrichtung

Die Himmelsrichtungen werden auf einen kompletten Kreis von $360°$ aufgeteilt. Norden liegt auf $0°$ respektive $360°$, Osten auf $90°$, etc. Um eine Himmelsrichtung wiederholbar als Schlüsseleingabe zu nutzen, verwenden wir die vier Haupt- und die vier Nebenhimmelsrichtungen. Dabei wird eine Himmelsrichtung bestimmt, wenn sich die gemessene Richtungsangabe des Smartphones im Bereich von $\pm 22.5°$ der betreffenden Himmelsrichtung befindet. Also z.B. für Norden, wenn sich die Richtungsangabe im Bereich zwischen $337.5°$ und $360°$ oder zwischen $0°$ und $22.5°$ befindet (wie in Abbildung 5.22 dargestellt). Eine detaillierte Aufstellung der einzelnen Bereiche für jede der acht Himmelsrichtungen ist im Anhang B.4.1 zu finden.

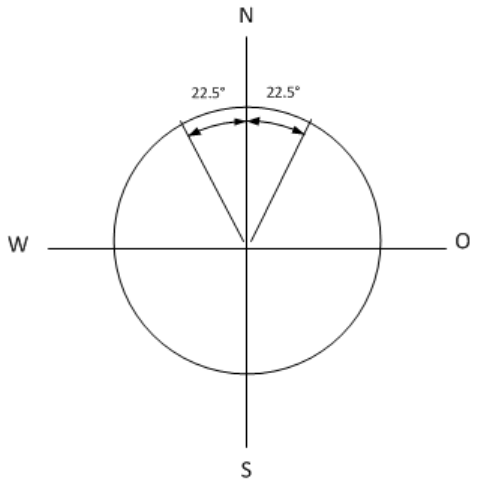


Abbildung 5.22.: Aufteilung der Himmelsrichtungen für Norden

6. Umsetzung

6.1. Schnittstelle auf die KryptonIT Bibliothek

Die Fassade hat drei verschiedene Verantwortlichkeiten und besteht im Wesentlichen aus dem Daten Modell, der KryptonIT Fassade und Funktionalität zum Austauschen (Import/Export) von Stores.

Die KryptonIT Fassade stellt sicher, dass der Zugang zur KryptonIT Library zustandslos erfolgt und abstrahiert dabei Anwendung der KryptonIT Bilbiothek auf das Nötigste (siehe auch Kapitel 4.2). Somit verkleinert sie auch den Funktionsumfang der Bibliothek auf genau das Minimum, welches in unseren Applikation benötigt wird. Für jeden der benötigten Schritte wurden in der Klasse `StoreManager` eine statische Methode definiert. Diese Methoden werden zusammen mit dem Daten Modell für die Erstellung und Abfrage von Stores verwendet.

Die beiden Klassen `SecretConverter` und `NumberBaseConverter` dienen zum Konvertieren von Zahlen in Text-Geheimnisse und umgekehrt. Die Klassen werden als Werkzeuge zur Verfügung gestellt. Die Benutzer der Fassade sind aber selbst verantwortlich für die Konversion (je nach Store Geheimnis Typ).

Das Daten Modell (im Package `ch.bfh.kryptonIT.facade.data`) besteht aus mehreren Klassen, welche dazu verwendet werden, die Zwischenresultate der einzelnen Schritte zusammenzufassen. Die Klassen sind reine Daten-Behälter und enthalten keine Logik. Aufgrund von Limitationen in der KryptonIT Bibliothek, mussten wir eine eigene Implementation vom Interface `Space` erstellen. Die Klasse `SimpleKrypteXSpace` ist ein reiner Datenbehälter und führt keine Validierung die Eingabe-Parameter durch (im Gegensatz zu der Klasse `KrypteXSpace` aus der KryptonIT Bibliothek).

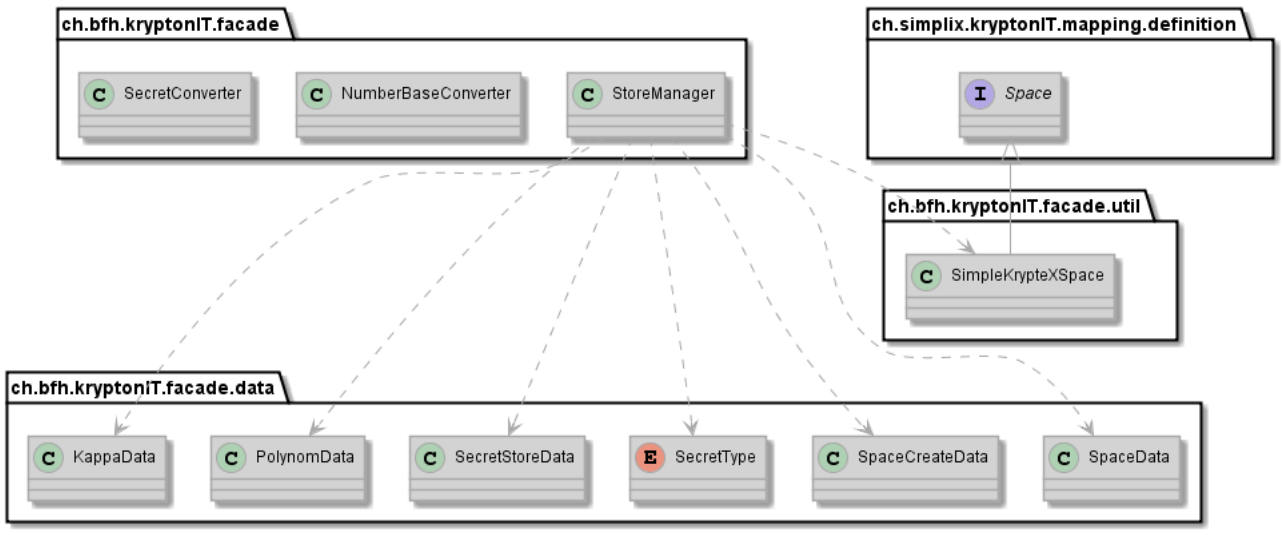


Abbildung 6.1.: Klassendiagramm der Fassade mit den Abhängigkeiten zu den Modell Klassen.

Für die grössen-optimierte Speicherung oder den manuellen Austausch von Inhalten aus den Modell Klassen kann die JSON Serialisierung verwendet werden. Diese verwendet die org.json Bibliothek [53], um möglichst kurze JSON Zeichenfolgen zu erstellen und umgekehrt wieder ein Objekt aus einer JSON Zeichenfolge zu erstellen. Um die Grösse eines Stores für den Austausch weiter zu optimieren, wurde die GZip Funktionalität[1] von Java verwendet. Ob ein Store für den Austausch Gezipt oder in Klartext als JSON weitergegeben wird, ist dem Benutzer der Fassade überlassen (siehe Abbildung 6.1).

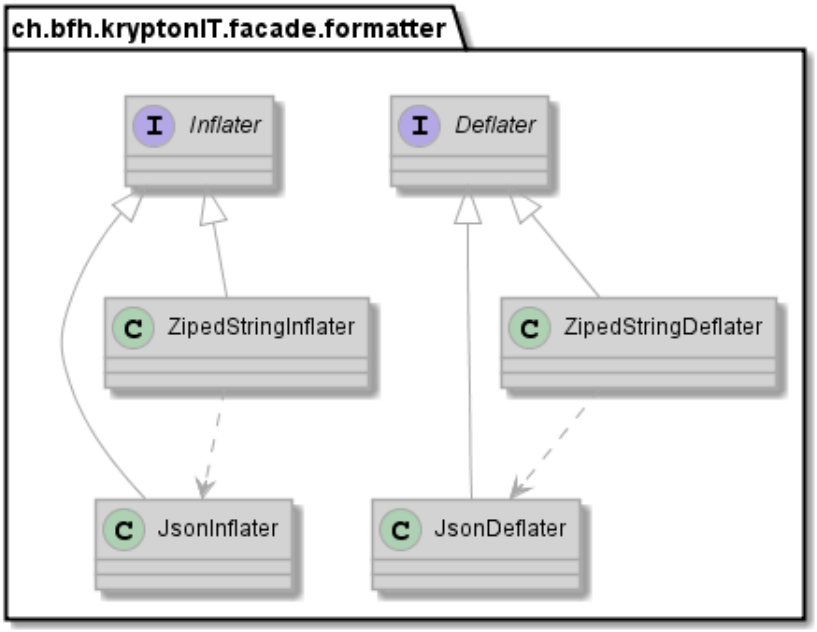


Abbildung 6.2.: Klassendiagramm der Fassaden-Funktioanlität für Serialisierung und Deserialisierung.

[1] GZIPOutputStream, http://docs.oracle.com/javase/6/docs/api/java/util/zip/GZIPOutputStream.html

6.2. Store Sicherheits Stufen

Um verschiedene Sicherheits-Stufen abzubilden, wurden für die sicherheitsrelevanten Store Parameter (siehe Kapitel 4.2) „Sicherheitsparameter 1 & 2“, die Anzahl der zufälligen Punkte und die verwendete Hash-Methode drei Garnituren erstellt. Die drei Garnituren werden in die Stufen „Normal“, „Hoch“ und „Verrückt“ eingeteilt. Diese drei Stufen werden dem Benutzer sowohl in der Android, als auch in der GWT Applikation mit den selben Werten angeboten. In Anhang C.1 sind die Werte für alle drei Stufen definiert.

6.3. Modulare Exponentiation

Die modulare Exponentiation wurde als vierte Hash-Methode in die KryptonIT Bibliothek integriert. Sie kann als einzige Geheimnis-Räume grösser als 2^{128} verarbeiten, ohne dass die Methode unsicher wird. Der Algorithmus potenziert den Offset o mit dem Schlüssel key und das Resultat wird mit dem Geheimnis-Raum P modulo gerechnet, um x zu erhalten.

$$x = o^{key} \, mod \, P$$

Möchte man die Operation umkehren, müsste man den diskreten Logarithmus zur Basis o finden. Es wird angenommen [54], dass dieser schwer zu finden ist. Falls nicht, würden die Grundlagen der modernen Kryptographie widerlegt.

6.4. Web Applikation

6.4.1. Entwicklungsumgebung

Für die Umsetzung der Google Web Toolkit Applikation wurde die Java Entwicklungsumgebung Eclipse zusammen mit dem Google Web Toolkit SDK und der zugehörigen App Engine von Google verwendet. Weiter wurden die Webbrowser Mozilla Firefox und Chromium verwendet, um die Applikation laufen zu lassen. (siehe auch Anhang C.2.1)

6.4.2. Aufbau der Applikation

Paketdiagramm

Die Applikation ist in elf Java Pakages aufgeteilt.

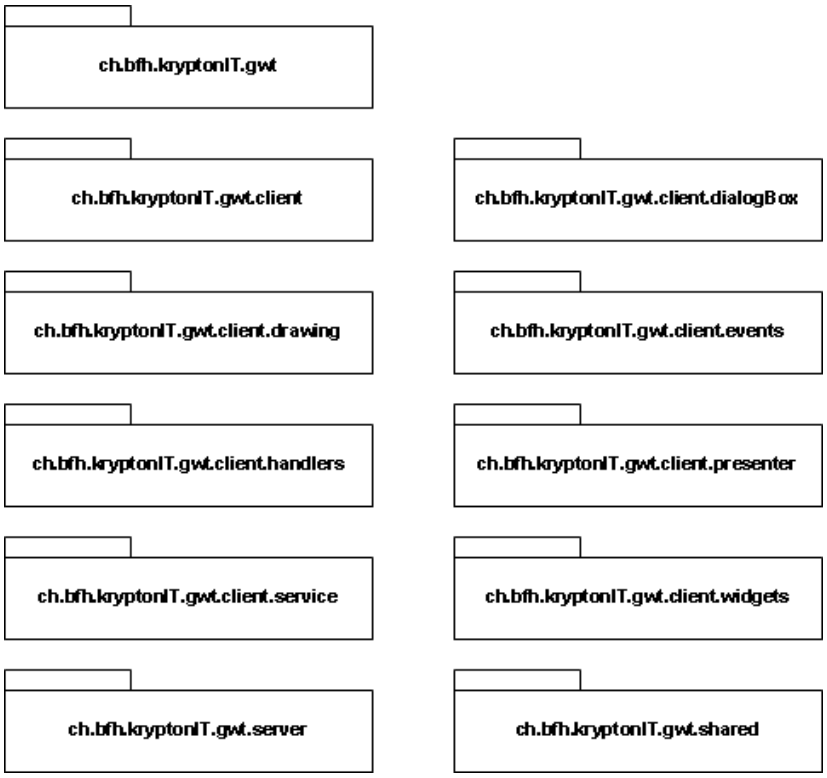


Abbildung 6.3.: GWT - Java Pakages

- ch.bfh.kryptonIT.gwt : Enthält die Datei KryptonIT.gwt.xml. Diese Datei ist die Modul-Datei der Anwendung. Sie definiert den Einstiegspunkt, die benötigten Module und Servlets und die verwendeten Sprache-Dateien.
- ch.bfh.kryptonIT.gwt.client : Enthält die Einstiegs-Klasse KryptonIT.java und die Klasse AppController.java.
- ch.bfh.kryptonIT.gwt.client.dialogBox : Enthält die erstellten DialogBox-Klassen.

- ch.bfh.kryptonIT.gwt.client.drawing : Enthält die Klassen CoordinateSystem.java und Axis.java, welche für den Zeichnungsbereich verwendet werden.
- ch.bfh.kryptonIT.gwt.client.events : Enthält die erstellten Event-Klassen.
- ch.bfh.kryptonIT.gwt.client.handlers : Enthält die erstellten Handler-Interface-Klassen.
- ch.bfh.kryptonIT.gwt.client.presenter : Enthält die erstellten Presenter-Klassen.
- ch.bfh.kryptonIT.gwt.client.service : Enthält die Klassen KryptonITService.java und KryptonITServiceAsync.java. Das sind die Interface-Klassen für den KryptonITService.
- ch.bfh.kryptonIT.gwt.client.widgets : Enthält die erstellten Widget-Klassen und die View-Klassen.
- ch.bfh.kryptonIT.gwt.server : Enthält die auf dem Server ausgeführte Klasse KryptonITServiceImpl.java. Das ist die Implementierung des erstellten KryptonITService.
- ch.bfh.kryptonIT.gwt.shared : Enthält die Klasse FieldVerifier.java und die Klasse Languages.java zusammen mit den erstellten Sprach-Dateien (.properties).

Klassendiagramm

Weil ein komplettes Klassendiagramm der Applikation für die Dokumentation viel zu gross und auch nicht mehr sehr übersichtlich ist, wird hier ein leicht vereinfachtes Klassendiagramm gezeigt. Die Klassen des KryptonITService wurden absichtlich weggelassen und werden im nächsten Kapitel separat erklärt.

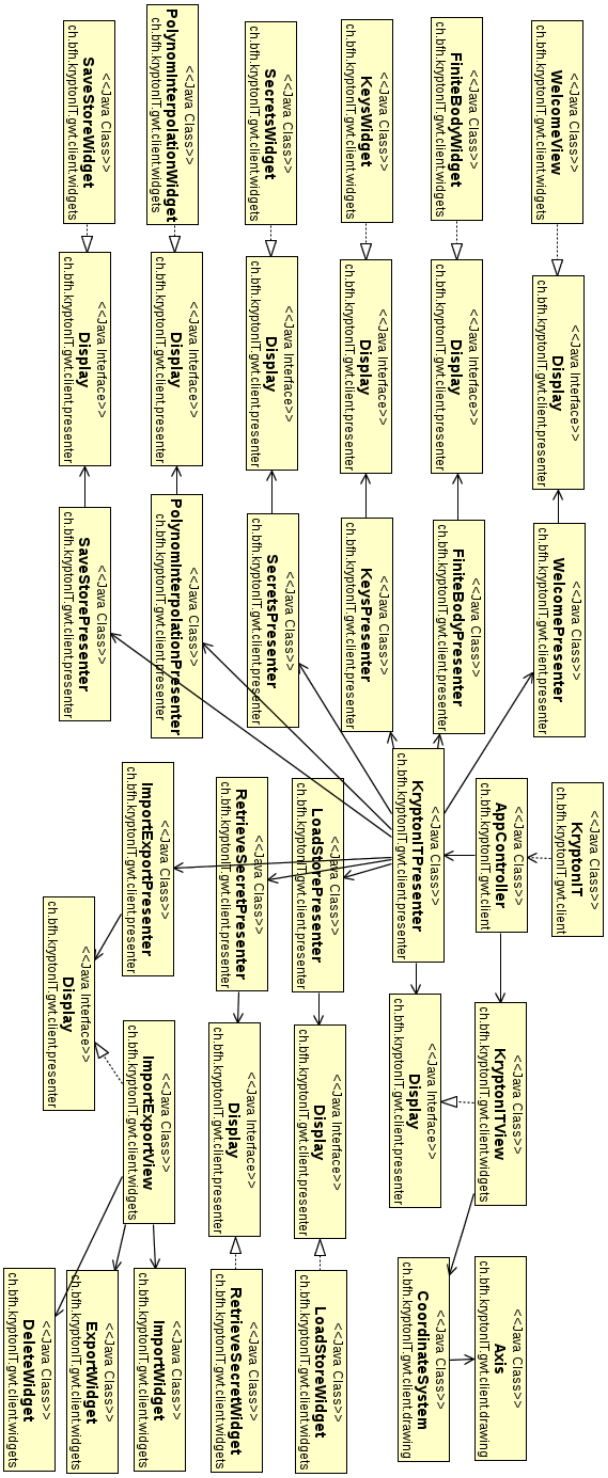


Abbildung 6.4.: Vereinfachtes Klassendiagramm der GWT-Applikation

6.4.3. Client-Server Kommunikation mit GWT Remote Procedure Call (RPC)

Mit dem integrierten GWT RPC Framework können die Komponenten einer Webapplikation Java-Objekte über HTTP zwischen Client und Server übertragen. [55] Der serverseitige Code, welcher vom Client aufgerufen wird, wird häufig als Service bezeichnet. Die Umsetzung des GWT RPC Service basiert auf der Java Servlet Architektur.

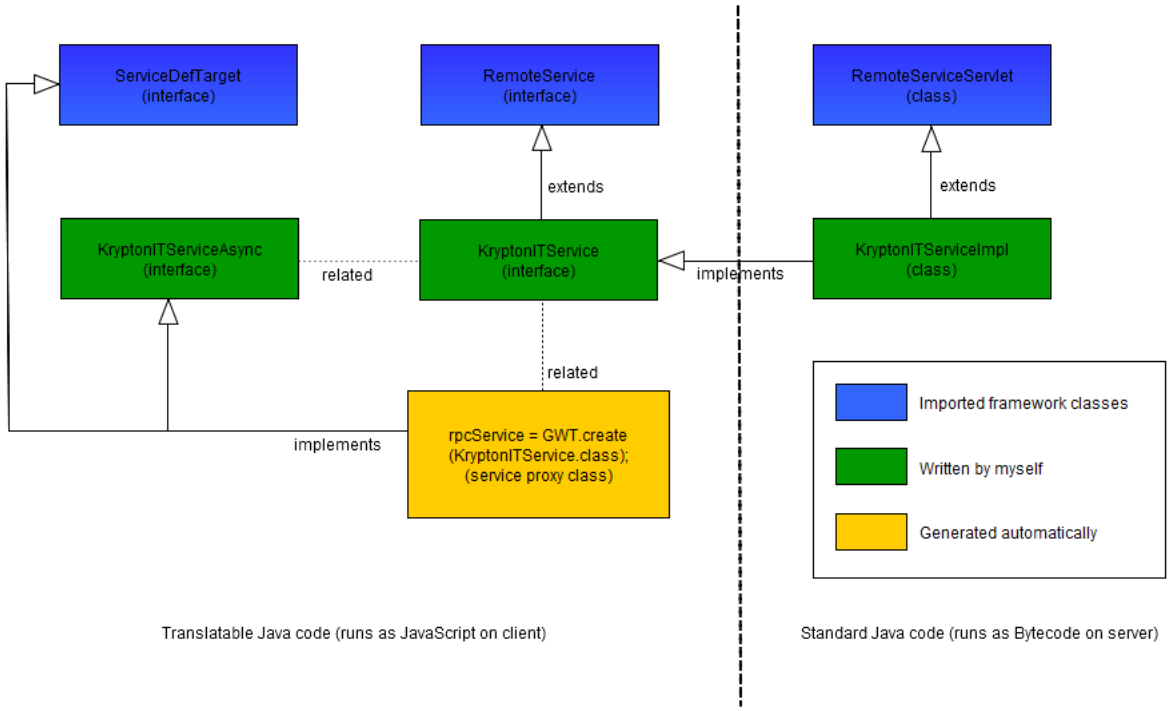


Abbildung 6.5.: Java Komponenten des KryptonIT Service

KryptonIT-Service

Für den KryptonIT-Service (siehe Abbildung 6.5), mussten drei Komponenten erstellt werden:

- Ein Interface (KryptonITService.java), welches von der Klasse RemoteService abstammt und alle RPC Methoden auflistet, welche der Service zur Verfügung stellt. (siehe Anhang C.2.2)
- Eine Klasse (KryptonITServiceImpl.java), welche von der Klasse RemoteServiceServlet abstammt und das gerade erstellte Interface implementiert.
- Ein asynchrones Interface (KryptonITServiceAsync.java), um den Service vom clientseitigen Code aufrufen zu können.

Im Client Code wird eine automatisch generierte Proxy Klasse benutzt, um mit dem Service sprechen zu können. GWT handelt die Serialisierung der Java Objekte ab, welche an den Client zurückgegeben werden und deserialsiert die vom Client an den Server übergebenen Parameter in den Methodenaufrufen.

```
KryptonITServiceAsync rpcService = GWT.create(KryptonITService.class);
rpcService.getSomething([parmeters], new AsyncCallback<[returnType]>() {
        public void onFailure(Throwable caught) {
                // doSomething
        }

        public void onSuccess([returnType] result) {
                // doSomething
        }
});
```

6.4.4. Presenter

Für die Applikation wurden zehn Presenter-Klassen erstellt, welche die Anwendungs-Logik beinhalten. Jeder Presenter ist für die Funktionalität einer View- oder Widget-Klasse zuständig. (siehe Anhang C.2.3) Die Presenter kommunizieren miteinander über Events (siehe Kapitel 6.4.8) und können so einander die aktualisierten Parameter übergeben.

6.4.5. Views

Die Anwendung wurde in drei Views aufgeteilt.

- KryptonITView : Die Main-View der Applikation. Sie beinhaltet den Haupttitel und ein TabLayoutPanel (siehe Kapitel 6.4.10) mit 4 Tabs. Der erste Tab beinhaltet die WelcomeView. Der zweite Tab beinhaltet ein StackLayoutPanel (siehe Kapitel 6.4.10) als Container für die Widgets, den Zeichnungsbereich (DrawingArea) und den Informationsbereich, welche für die Erstellung eines Stores benötigt werden. Der dritte Tab beinhaltet ein StackLayoutPanel als Container für die Widgets, den Zeichnungsbereich (DrawingArea) und den Informationsbereich, welche für das Abfragen eines Stores benötigt werden.
- WelcomeView : Beinhaltet einen Begrüssungstext, das Logo und einen Einleitungstext.
- ImportExportView : Beinhaltet die Widgets, welche für das Importieren, Exportieren und Löschen eines Stores benötigt werden.

6.4.6. Eigene Widgets

Die vielen Eingabe-, Ausgabe-Felder und Knöpfe wurden pro Eingabe-Schritt gruppiert. Die Gruppen von diesen Feldern wurden als eigene Widgets implementiert, um sie in separate Klassen auslagern zu können. Die eigenen Widgets sind also keine „neuen“ Widgets, sondern nur selbst erstellte Gruppierungen von bestehenden Widgets in eigenen Klassen. (siehe Anhang C.2.4)

6.4.7. Dialog Boxen für Hilfe Fenster

Für die Darstellung der im Kapitel 4.8.5 erwähnten Hilfe Fenster, wurden 12 eigene DialogBox-Klassen erstellt. Jede DialogBox beinhaltet alle Hilfe-Texte eines Hilfe-Fensters.

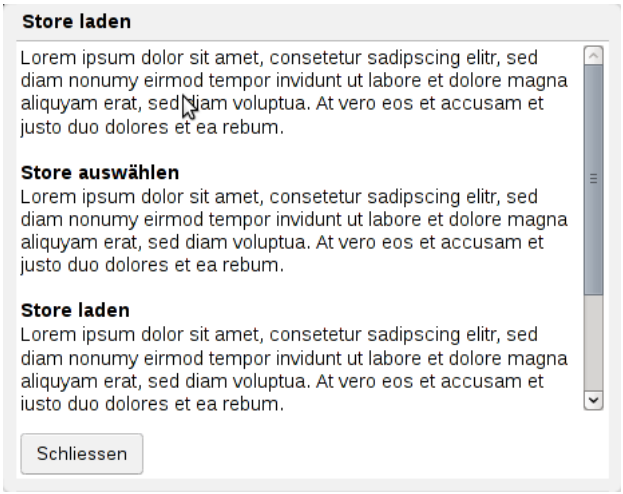


Abbildung 6.6.: Hilfe Fenster zum Schritt „Finiter Körper“

6.4.8. Events

Für die Übergabe der Parameter und Kommunikation zwischen den einzelnen Klassen wurden Event- und Handler-Klassen erstellt. (siehe Anhang C.2.5) Die einzelnen Presenter der Applikation benutzen die Events, um den KryptonITPresenter über den aktuellen Verlauf eines Durchführungsschrittes zu informieren. Der KryptonITPresenter kann anhand der aufgefangenen Events die grafische Ansicht (Zeichnungsbereich) und den Informationsbereich aktualisieren.

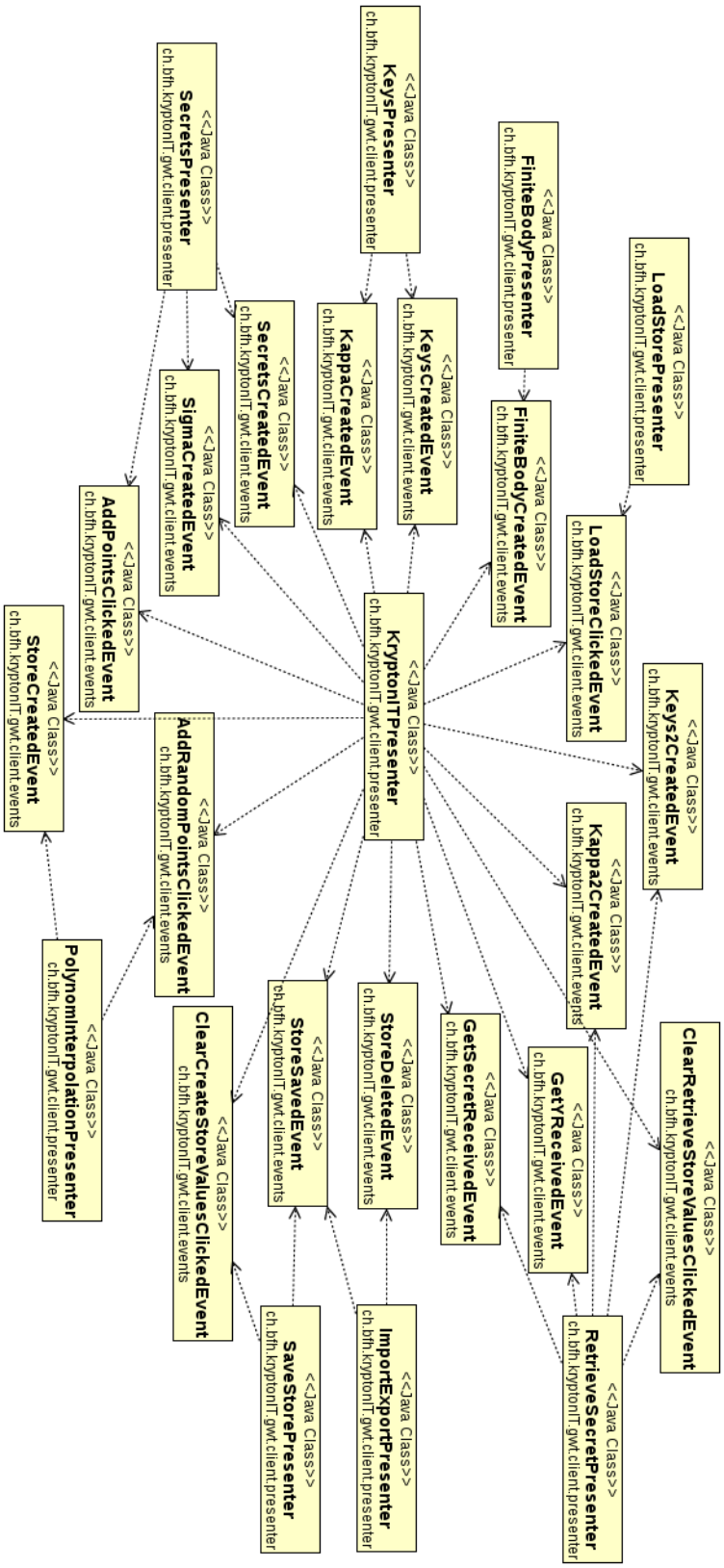


Abbildung 6.7.: Verbindungen zwischen Presenter und Events

6.4.9. Generierung der Zufallspunkte

In der KryptonIT Bibliothek von Reto König ist das generieren der Zufallspunkte nicht als separate Methode implementiert. Sie werden erst ganz am Schluss, bei der createStore-Methode() generiert, bei welcher die Anzahl der zu generierenden Punkte mitgegeben wird. Die Punkte werden auch nicht zurückgegeben. Dies ist nicht sehr praktisch, da in der GWT Applikation die generierten Zufallspunkte ausgegeben werden sollen. Darum wurde entschieden, keine Zufallspunkte über die createStore()-Methode zu generieren, sondern selbst in einem früheren Schritt als der Store Erstellung, bereits eine definierbare Anzahl Zufallspunkte zu generieren, welche der createStore()-Methode als „normale“ zusätzliche Punkte (wie Schlüssel-und Geheimnis Paare) übergeben werden. Die x- und y-Werte für die Zufallspunkte werden mit der random()-Methode von der Klasse java.lang.Math berechnet.

6.4.10. Benutzeroberfläche/Design

Layout mit LayoutPanels

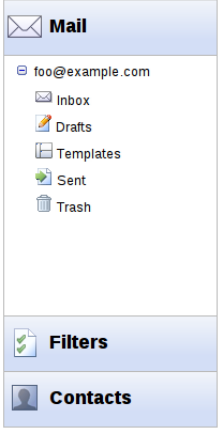
GWT 2.0 bietet eine neue Form von Panels (LayoutPanels) an, welche einfach animiert werden können. [56] Um diesen Vorteil ausnutzen zu können, werden diese neuen Komponenten in der GWT Applikation eingesetzt.

Für die Tab-Darstellung, welche die vier Seiten der Applikation beinhaltet, wird das TabLayoutPanel verwendet.

[57]

Abbildung 6.8.: TabLayoutPanel Widget von GWT

Für die Aufteilung der einzelnen Schritte in Akkordeon-Form beim Erstellen und beim Abfragen eines Stores, wird das StackLayoutPanel eingesetzt.



[58]

Abbildung 6.9.: StackLayoutPanel Widget von GWT

ScrollPanel

Da Standarmässig in den neuen LayoutPanels nicht gescrollt werden kann, wurde für die Bereiche innerhalb der LayoutPanel jeweils ein ScrollPanel implementiert, um das Scrollen darin zu ermöglichen.

FlowPanel anstatt HorizontalPanel und VerticalPanel

Die Inhalte im TabLayoutPanel und im StackLayoutPanel werden in FlowPanels abgefüllt. FlowPanels rendern im Vergleich zu Horizontal- und VerticalPanel ein -Tag als Container und nicht ein <table>-Tag. Die horizontale oder vertikale Anordung der FlowPanels wird mit Hilfe von CSS mit dem float Attribut gesteuert.

Darstellung mit CSS

Um die Applikation einheitlich gestalten zu können, wurde das Design mit Cascading Style Sheets (CSS) realisiert. Hierfür wurde bei jedem Element, das formatiert werden musste, das Attribut addStyleName("[Name der Style Klasse]") hinzugefügt. Im File KryptonIT.css sind die Style Klassen mit ihren Attributen definiert.

6.4.11. Zeichnungsbereich

Im Zeichnungsbereich werden grundsätzlich vier Achsen dargestellt, zwei davon bilden ein Koordinatensystem. Diese logische Gruppierung wurde beim Aufbau der Struktur übernommen. Es wurde die Klasse „Axis“ für eine Achse erstellt und die Klasse „CoordinateSystem“ für das Koordinatensystem. Sie beinhaltet vier Instanzen der Klasse „Axis“. Zwei, um das Koordinatensystem zu bilden und zwei Weitere für die Schlüssel- und die Geheimnis-Achse.

Für die Darstellung der Werte-Achsen werden Linien gezeichnet.

```
Line line = new Line(x1, y1, x2, y2);
```

Für die Beschriftung wird Text benutzt.

```
Text axisText = new Text(x, y, axisName);
```

Für Darstellung der Punkte auf den Achsen und im Koordinatensystem werden Kreise gezeichnet.

```
Circle circle = new Circle(x, y, 6);
```

Benötigte Umrechnungen im SVG-Bereich

Im Zeichnungsbereich befindet sich der Nullpunkt in der oberen linken Ecke. Beim Koordinatensystem, das gezeichnet werden soll, befindet sich der Nullpunkt unten links. Dies erfordert, dass alle Werte der y-Koordinaten umgerechnet werden müssen.

Weiter befindet sich um das Koordinatensystem herum ein Abstand, welcher beim Zeichnen der Achsen und beim Einzeichen von Punkten auf den Achsen und im Koordinaten System ebenfalls berücksichtigt werden muss.

Da die Grösse der gezeichneten Grafik immer dieselbe ist, die Werte der Achsen aber bei erneutem Erstellen oder Abfragen eines Stores variieren können, werden zusätzlich jeweils die anzuzeigenden Werte in die benötigte Grösse für die Darstellung umgerechnet.

Beschriftung der Achsen und der Punkte

Die Zahlenwerte der Schlüssel und Geheimnisse, sowie die transformierten x und y Werte sind aufgrund der gewählten Umwandlungs- und Verschlüsselungsverfahren sehr gross. Da dies für die Beschriftung der Achsen problematisch ist, wurde entschieden, die Zahlen der Beschriftung in der wissenschaftlichen Darstellung anzuzeigen. Das folgende Format wurde für die wissenschaftliche Darstellung verwendet:

```
formattedxAxisValue = NumberFormat.getFormat("#.#E0").format(xAxisValue);
```

Klickt man mit der Maus auf die Achse, wird einem der Zahlenwert in der normalen Darstellung angezeigt.

Die gesetzten Punkte auf den Achsen und im Koordinatensystem haben standardmässig keine Textanzeige. Klickt man mit der Maus über einen der Punkte, wird einem auch hier eine Anzeige mit dem Wert des Punktes eingeblendet.

6.4.12. Informationsbereich

Der Informationsbereich stellt die textliche Darstellung der berechneten Werte dar. Bei jedem Durchgeführten Schritt beim Erstellen oder Abfragen eines Stores, wird ein weiteres DisclosurePanel eingeblendet. Das DisclosurePanel wurde für die Darstellung des Inhalts gewählt, weil die Informationen darin auf- und zugeklappt werden können. So wird bei der Ausgabe Platz gespart und es können nur die gewünschten Informationen dargestellt werden.

Abbildung 6.10.: DisclosurePanel Widget von GWT zugeklappt

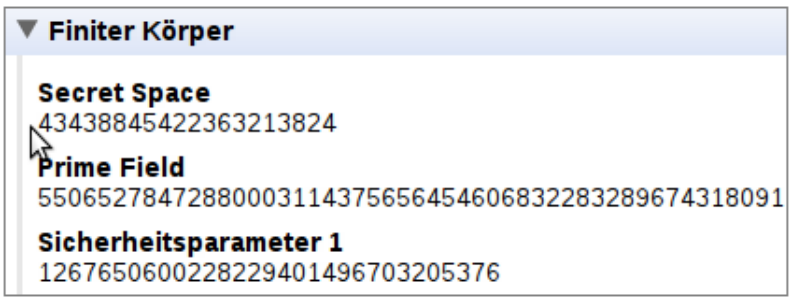


Abbildung 6.11.: DisclosurePanel Widget von GWT aufgeklappt

6.4.13. Loading

Weil der erste Aufruf der Applikation etwas länger dauern kann, wurde auf der Startseite eine Loading-Anzeige eingebaut, welche angezeigt wird, bis die Applikation fertig geladen ist.

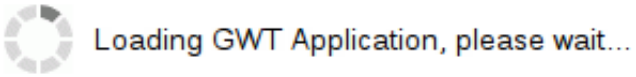


Abbildung 6.12.: Loading-Anzeige während dem Laden der Applikation

Folgender Code wurde dafür in der HTML-Seite KryptonIT.html eingebaut:

```
<div id="loadingMessage" style="position:absolute;left:50\%;top:50\%;margin-
    left:-11em;margin-top:-5em;">
   <div style="float:left;margin-right:10px;"><img src="images/loaderb32.gif"
       alt="loading"></div>
   <div style="width:22em;margin-top:8px;">Loading GWT Application, please
       wait...</div>
</div>
```

6.4.14. JavaScript deaktiviert

Diese Web Applikation funktioniert nur, wenn JavaScript im Browser aktiviert ist. Für den Fall, dass der Benutzer in seinem Browser JavaScript deaktiviert haben sollte, wird folgende Meldung angezeigt.

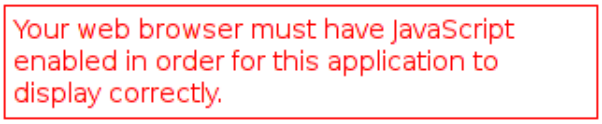


Abbildung 6.13.: Meldung, falls JavaScript nicht aktiviert ist.

Für die Prüfung, ob JavaScript deaktiviert ist, wurde in der HTML-Seite KryptonIT.html folgender Code eingebaut:

```
<noscript>
   <div style="width:22em;position:absolute;left:50\%;top:50\%;margin-left:-11
       em;margin-top:-5em;color:red;background-color:white;border:1px␣solid␣
       red;padding:4px;font-family:sans-serif">
       Your web browser must have JavaScript enabled in order for this
           application to display correctly.
   </div>
</noscript>
```

6.4.15. Slider

Da in den Standard-Widgets von GWT kein Slider-Widget angeboten wird, wurde mit Hilfe der Anleitung „Creating a GWT Wrapper for the JQuery UI Slider“ eine Umgehungslösung umgesetzt. [30]

6.4.16. Speichern und Laden eines Stores

Um das SecretStoreData Objekt, welches den Store darstellt, im Browser speichern oder laden zu können, wird als Erstes ein Storage Objekt erstellt, welches die Local Storage Daten des benutzten Browser beinhaltet.

```
Storage stockStore = Storage.getLocalStorageIfSupported();
```

Mit der setItem()-Methode des Objekts können weitere Stores in Textform im Local Storage des Browsers abgelegt werden.

```
stockStore.setItem(store.getName(), jsonString);
```

Um den Store in Text zu verwandeln, wird die getJsonString()-Methode des KryptonITService aufgerufen. Sie verwandelt den übergebenen Store (SecretStoreData Objekt) in einen JSON-String und gibt diesen zurück.

Um einen gespeicherten Store zu laden, wird die getItem()-Methode des Storage Objekts verwendet.

```
stockStore.getItem(key);
```

Der geladene Store-String wird mit der getStore()-Methode des KryptonITService wieder in ein SecretStoreData Objekt verwandelt und der Applikation übergeben.

6.4.17. Löschen eines Stores

Ein gespeicherter Store kann wieder gelöscht werden, indem der Store-String im Local-Storage des Browsers entfernt wird.

```
stockStore.removeItem(store);
```

6.4.18. Importieren eines Stores

Um einen Store zu importieren, wird der eingefügte JSON-String eines Stores zuerst auf Gültigkeit überprüft und dann, gleich wie beim Speichern eines Stores (siehe Kapitel 6.4.16), mit der setItem()-Methode des Storage Objekts im Local Storage des Browsers abgelegt.

```
stockStore.setItem(store.getName(), jsonString);
```

6.4.19. Exportieren eines Stores

Um einen Store als Text zu exportieren, wird gleich wie beim Speichern eines Stores (siehe Kapitel 6.4.16), die getJsonString()-Methode des KryptonITService aufgerufen. Sie verwandelt den übergebenen Store (SecretStoreData Objekt) in einen JSON-String und gibt diesen zurück.

Für das Exportieren eines Stores als QR-Code wird die getCompressedJsonString()-Methode des KryptonITService aufgerufen. Sie komprimiert einen Store, weil ein QR-Code nur eine beschränkte Grösse hat und liefert diesen als Base64 encodierten String zurück. Dieser String wird mit Hilfe der Google Chart Tools in einen QR-Code verwandelt. [59] Dafür wird die folgende URL aufgerufen, an welche der Store-String angehängt wird.

```
String url = "http://chart.apis.google.com/chart?cht=qr&chs=350x350&chld=L&
    choe=UTF-8&chl=" + URL.encodeQueryString(result);
```

Als Rückgabewert erhält man wieder eine URL. Diese URL wird aufgerufen und ein PopupPanel mit dem erstellten QR-Code wird eingeblendet. Ist der Store-String länger als 4296 Zeichen, was dem grössten QR-Code Inhalt entspricht der verwendet werden kann, wird ein PopupPanel ohne Inhalt angezeigt.

6.4.20. Validierungen

Damit der User die Reihenfolge der durchzuführenden Schritte einhält und keine falschen Eingaben machen kann, wurden 31 Validierungen in die Anwendung eingebaut. (siehe Anhang C.2.6)

6.4.21. Internationalisierung

Damit die Applikation in mehreren Sprachen angezeigt werden kann, wurden die angezeigten Texte internationalisiert. [60]

Folgende Schritte mussten für die Internationalisierung durchgeführt werden:

- Es wurde ein Interface Languages.java erstellt, in welchem die Methoden für jede Text-Variable definiert wurden.

  Bsp. String welcomeTitle();

- Es wurden drei .properties Dateien erstellt:

  - Die Datei Languages.properties beinhaltet die Default Werte der Text-Variablen auf Englisch, wenn die Text-Variablen in einer Sprache nicht vorhanden sind.

  - Die Datei Languages_en.properties beinhaltet die Werte der Text-Variablen auf Englisch.

  - Die Datei Languages_de.properties beinhaltet die Werte der Text-Variablen auf Deutsch.

    Bsp. welcomeTitle=Willkommen

- Im Modul-File KryptonIT.gwt.xml wurden die folgenden Zeilen eingetragen:

```
<extend-property name="locale" values="de" />
<extend-property name="locale" values="en" />
<set-property-fallback name="locale" value="en" />
```

- Um die Text-Variablen zu verwenden, muss im Java Code die Datei Languages.class (wurde vom System automatisch erstellt) instanziert werden. Das erstellte Objekt kann danach die Methoden des Interface Languages.java verwenden.

```
Languages language = GWT.create(Languages.class);
language.welcomeTitle();
```

6.5. Mobile Applikation

6.5.1. Entwicklungsumgebung

Für die Umsetzung der Android Applikation wurde die Java Entwicklungsumgebung Eclipse zusammen mit dem Android SDK verwendet. Google stellt das Plugin ADT für Eclipse zur Verfügung, welches bequemes Arbeiten mit den Android SDK Werkzeugen über eine grafische Benutzeroberfläche erlaubt. Eine detaillierte Liste der verwendeten Software kann im Anhang C.3.1 gefunden werden.

6.5.2. Überblick

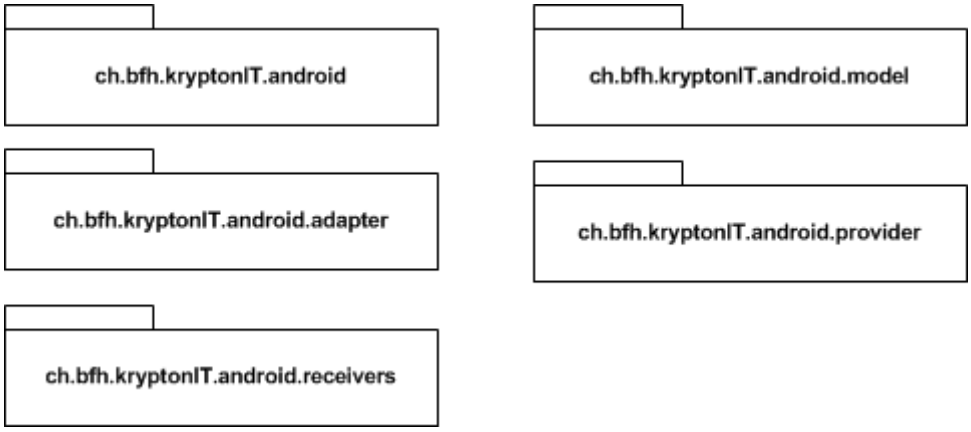


Abbildung 6.14.: Android - Java Pakages.

Ein kompletter Überblick über die Klassen und deren Zusammenhänge ist in Anhang C.3.3 zu finden. Die Applikation ist in folgende fünf Java Pakages aufgeteilt.

`ch.bfh.kryptonIT.android`: siehe Kapitel 6.5.4

`ch.bfh.kryptonIT.android.model`: siehe Kapitel 6.5.3

`ch.bfh.kryptonIT.android.adapter`: siehe Kapitel 6.5.5

`ch.bfh.kryptonIT.android.provider`: siehe Kapitel 6.5.7

`ch.bfh.kryptonIT.android.receivers`: Enthält einzig einen Intent Receiver für das Empfangen von Wifi Scan Resultaten.

6.5.3. Applikations-Modell

Da die Android Applikation spezifische Bedürfnisse an das Applikations-Modell stellt und damit die Daten von der Logik getrennt sind, haben wir uns entschieden, das Modell der Fassade zu ergänzen. Wenn möglich, werden die Fassade-Klassen innerhalb des Applikations-Modell gekapselt. Die in Abbildung 6.15 gezeigten Klassen repräsentieren die Artefakte, welche in der Applikation angezeigt oder verwaltet werden.

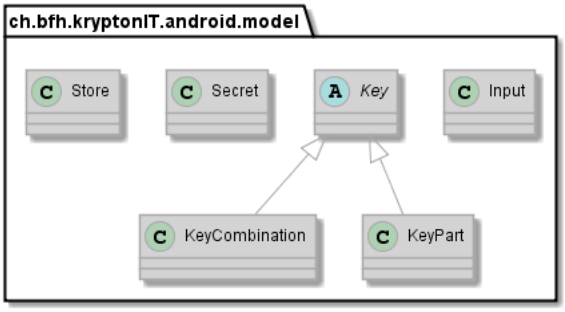


Abbildung 6.15.: Android - Modell Klassen.

6.5.4. Activities

Wie im Kapitel 4.9.4 definiert, wurde für jede Bildschirm-Maske eine separate Activity erstellt. Die Activities beinhalten die Applikations-Logik und interagieren mit den anderen Komponenten (z.B. das Starten anderer Activities oder das Lesen vom Content Provider). Die Abstrakte Klasse `MenuActivity` stellt ein Hilfe-Menü und die benötigte Logik, um die Hilfe anzuzeigen, zur Verfügung.

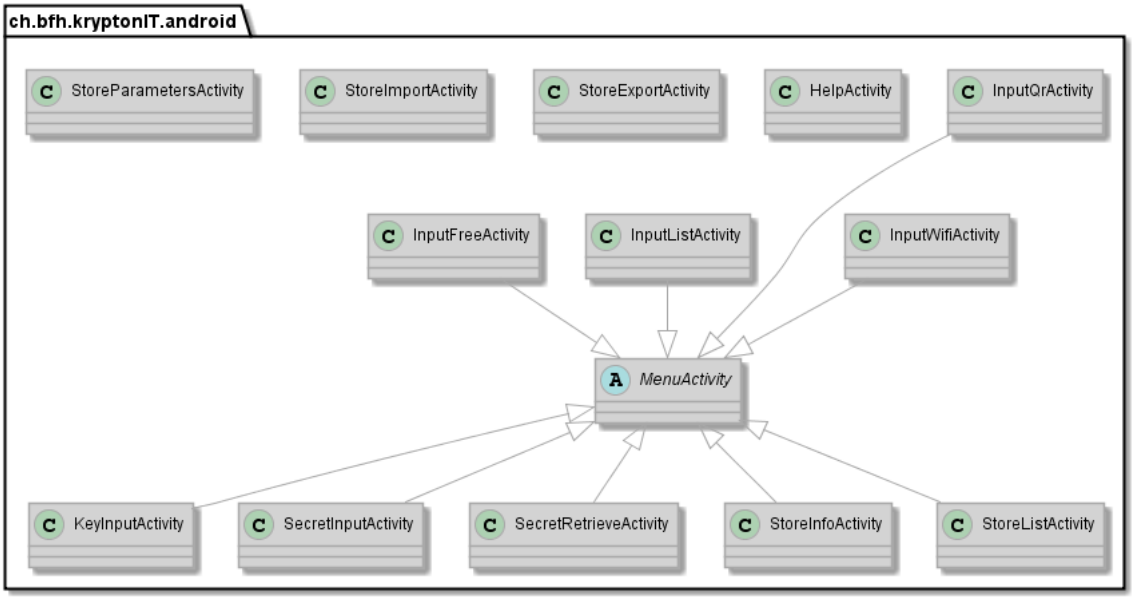


Abbildung 6.16.: Android - Activity Klassen.

Wie schon erwähnt, besteht jede Activity nur aus Logik, welche die Benutzeroberfläche und das Modell steuert. Das Aussehen und Design der Benutzeroberfläche ist in separaten Layout XML-Dateien definiert (siehe 6.5.6) und wird durch die Activity als Erstes beim Erstellen geladen. Danach werden, falls nötig die per Intent erhaltenen Parameter ausgewertet und die Ereignis-Routinen (Eventhandler) mit den UI Elementen eingerichtet. In Abbildung 6.17 wird dieser Vorgang in Pseudocode verdeutlicht.

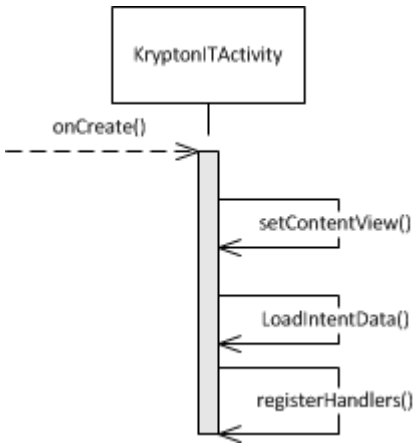


Abbildung 6.17.: Android - Activity Erstellung.

Hilfe

In die Applikation wurde eine Hilfe-Funktion integriert, aber ohne Inhalte dazu einzupflegen. Dazu wurde die Komponente `HelpActivity` geschaffen. Diese zeigt HTML-Seiten in einem Web-Browser Steuerelement an. Die HTML Seiten werden aus den Applikations-Ressourcen geladen und können somit auch Offline verwendet werden. Alle Activities, die von `MenuActivity` ableiten, können durch aufrufen der Methode `setHelpPage` die gewünschte Seite angeben, welche angezeigt werden soll. Ist dies nicht möglich (z.B. weil eine andere Basis-Klasse verwendet werden muss), kann die `HelpActivity` per Intent angesprochen werden.

```
Intent intent = new Intent(getApplicationContext(), HelpActivity.class);
intent.putExtra(Intents.EXTRA_KEY_HELPPAGE,
"file:///android_asset/test.html");
startActivity(intent);
```

Solange alle Hilfe-Seiten untereinander relativ verlinkt sind, kann der Benutzer innerhalb des Hilfe-Systems navigieren und gelangt zurück zur Ursprungs-Maske, wenn er den Android Zurück-Knopf verwendet.

6.5.5. Adapter

Jedes Element des Modells (siehe Kapitel 6.5.3) wird auf einer Maske in der Listen-Form angezeigt. Die Auflistung von Elementen passiert mit der Android Klasse `ListView` (oder Spezialisierungen von dieser). Um eine Liste von Elementen in einem `ListView` darzustellen, werden Adapter verwendet. In Abbildung 6.18 sind alle Activities mit der Abhängigkeit auf den entsprechenden Adapter aufgeführt. Weiter ist auch sichtbar, welchen Typ der Adapter auflistet. Der Adapter ist dafür zuständig, die Informationen aus dem Modell in ein Listen-Element zu laden. Dafür wird er vom Android System pro Element aufgerufen.

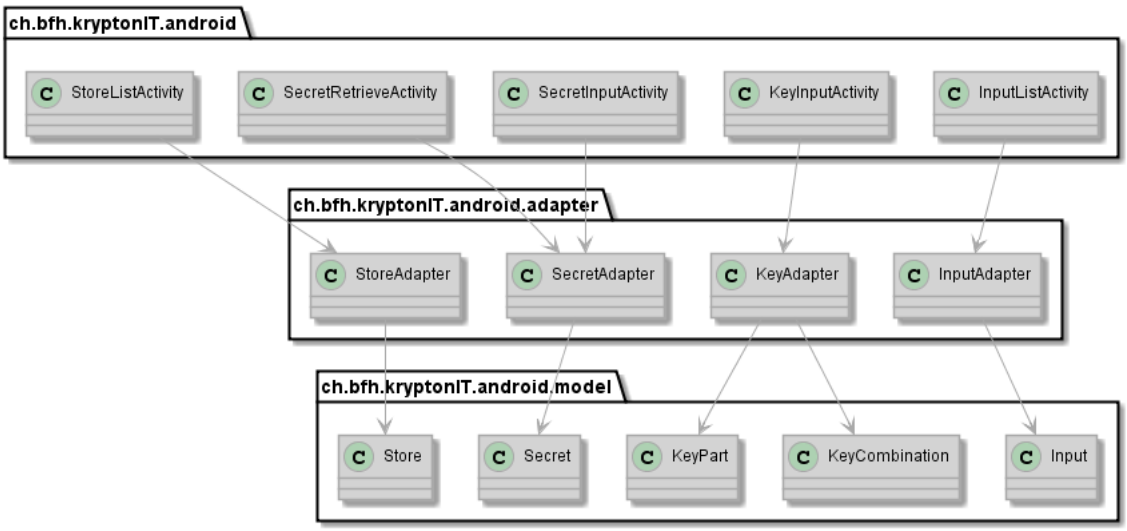


Abbildung 6.18.: Android - List Adapter.

6.5.6. Ressourcen

Alle Benutzeroberflächen sind in XML Layout Dateien definiert. Dabei wurde im Besonderen darauf geachtet, dass die Dimensionierung der einzelnen UI Elemente relativ erfolgt. So soll sichergestellt werden, dass die Benutzeroberfläche auf Geräten mit verschiedenen Bildschirmgrössen und Pixel-Dichten verwendbar sind. Als Beispiel ist in Abbildung 6.19 eine Maske der Applikation auf einem grossen 10.1 Zoll Bildschirm dargestellt.

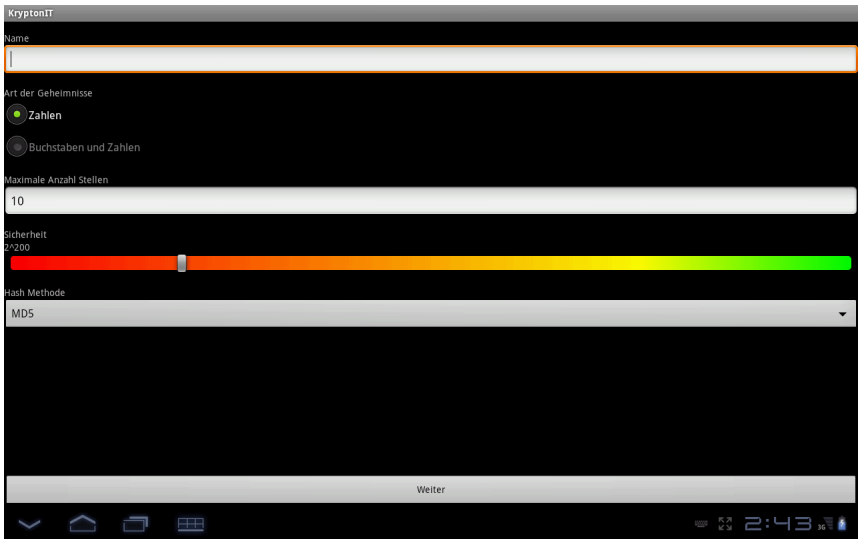


Abbildung 6.19.: Android - Benutzeroberfläche in WXGA (1280x768).

Alle Texte, welche in der Benutzeroberfläche verwendet werden, sind in Android XML Dateien abgelegt. Dieses Vorgehen wird von Google empfohlen und entspricht dem Internationalisierungs-Konzept von Android. Die Standard-Ressource ist in englischer Sprache gehalten und wird verwendet, wenn keine spezifische Sprach-Ressource zur Verfügung steht. Als spezifische Sprach-Ressourcen wird die Sprache Deutsch angeboten, welche geladen wird, wenn die Sprach-Einstellung auf dem Gerät entsprechend eingestellt ist.

6.5.7. Persistenz

Für das Speichern von Stores wurde ein Content Provider umgesetzt, welcher die Daten in einer SQLite Datenbank abspeichert. Die Speicherung des Polynoms wurde stark vereinfacht. Dieses wird bei der Speicherung in eine JSON Zeichenfolge serialisiert. Umgekehrt muss beim Laden eines Stores diese JSON Zeichenfolge wieder in eine Objekt Repräsentation umgewandelt werden.

Das Datenbank Schema der SQLite Datenbank wurde eins zu eins aus der Definition eines Stores in der Fassade übernommen. Für die Speicherung von BigInteger Datentypen, wird der Datentyp `BLOB` verwendet. Da Ganzzahl Datentypen in SQLite eine maximale Länge von 64 Bit haben, sind Sie für unsere Zwecke nicht geeignet. Die Speicherung eines BigInteger wird mit der Methode `toByteArray()` erledigt, welche eine Binär-Repräsentation der Zahl zurück gibt. Umgekehrt besitzt BigInteger einen Konstruktor, um aus dem Byte-Array eine Instanz der Zahl zu erhalten.

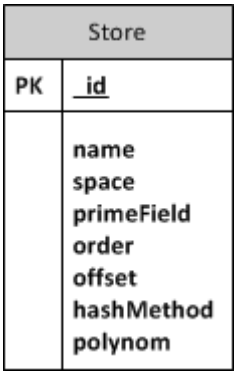


Abbildung 6.20.: Android SQLite ERD.

Der Zugriff auf gespeicherte Stores ist für andere Applikationen ausschliesslich über den Content Provider `SecretStoreProvider` möglich. Dieser wird über das Manifest der Android Applikation registriert und kann über die Authority `ch.bfh.kryptonIT.provider.SecretStore` gefunden werden.

Um alle gespeicherten Stores abzufragen, kann folgende Uri verwendet werden:
`content://ch.bfh.kryptonIT.provider.SecretStore/stores`

Für einen einzelnen Store wird die Content Uri mit der ID des Stores ergänzt. Hier wird als Beispiel der Store mit der ID 1 abgefragt:
`content://ch.bfh.kryptonIT.provider.SecretStore/stores/1`

6.5.8. Asynchrone Operationen

Um die Applikation nicht zu blockieren und einen ANR zu riskieren, wurde an zwei verschiedenen Stellen ein `AsyncTask` eingesetzt (siehe 4.9.6).

StoreParameterActivity: während der Entwicklung wurde beobachtet, dass das Berechnen des Secret Space und der Primzahl (für den finiten Körper) relativ lange dauert. Um die Applikation auf Geräten mit leistungsschwächerer Hardware einzusetzen, wird die Berechnung in einem `AsyncTask` durchgeführt.

SecretInputActivity: diese Activity erstellt den Store, wenn der Benutzer auf die Schaltfläche „Abschliessen“ drückt. Bei diesem Schritt wird die Polynominterpolation durchgeführt, welches je nach gewählten Parametern und der Anzahl von Key-Secret Paaren aufwändig ist.

6.5.9. Schlüssel Eingaben

Der Benutzer kann aus vier verschiedenen Eingabe-Möglichkeiten wählen, um einen Schlüssel einzugeben.

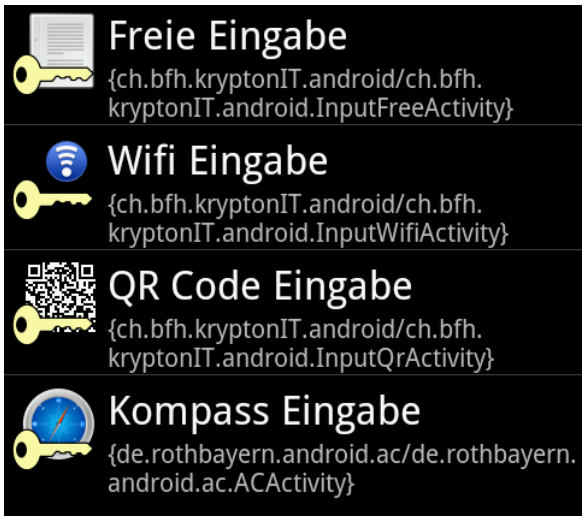


Abbildung 6.21.: Android Applikation - Wahl des Eingabe-Typs.

Die **Freie Eingabe** stellt die allgemeinste Form einer Schlüssel Eingabe dar. Der Benutzer kann eine beliebige Zeichenkette eingeben, welche zusammen mit dem Präfix „text:“ als Schlüssel verwendet wird.

Die **Wifi Eingabe** bezieht den Schlüssel-Wert (wie in Kapitel 5.2.2 beschrieben) aus dem Kontext und wird mit dem Präfix „wifi:“ versehen. Dieser Eingabe-Typ erlaubt es auch, eine korrekt formatierte BSSID einzugeben, falls das entsprechende Wifi im Benutzer-Kontext nicht vorhanden ist.

Für die automatisierte Text-Eingabe steht die **QR Code Eingabe** zur Verfügung. Bei der Wahl dieses Typ's wird die Barcode Applikation von ZXing [50] aufgerufen, um den Text-Inhalt des QR-Codes als Schlüssel-Wert zu verwenden. Damit diese Eingabe genutzt werden kann, muss die erwähnte Applikation auf dem Gerät installiert sein (siehe auch Kapitel 6.5.11).

Als letzten Eingabe Typ haben wir die **Kompass Eingabe** umgesetzt. Diese kann verwendet werden um eine der 8 Himmelsrichtungen (siehe Kapitel 5.2.2) aufzuzeichnen. Um die Visualisierung eines Kompass und die Ansteuerung der Sensoren nicht selbst implementieren zu müssen, haben wir die Applikation „Analog Compass“ [61] verwendet. Diese wurde unter der GPL veröffentlicht, und kann somit erweitert werden. Damit wir nicht gezwungen sind, die Android KryptonIT Applikation unter eine GPL kompatible Lizenz zu stellen, haben wir die „Analog Compass“ Applikation geringfügig erweitert. Die Erweiterung erlaubt, dass der Kompass von anderen Applikationen aufgerufen werden kann, um eine der 8 Himmelsrichtungen abzufragen. Die angepasste Kompass-Applikation ist selbstverständlich wiederum unter der GPL lizenziert und muss auf dem betreffenden Android System installiert sein, ansonsten wird eine Fehlermeldung angezeigt.

6.5.10. Validierungen

Hier wird darauf eingegangen, welche Eingabe-Validierungen vorgenommen wurden, um ein korrektes Funktionieren der Applikation und der darunterliegenden KryptonIT Methode zu gewährleisten.

- `StoreParameterActivity`
  - Der Store-Name darf nicht leer sein.
  - Die minimale Anzahl Zufallspunkte darf nicht grösser als die maximale Anzahl sein.
- `KeyInputActivity`
  - Ein Schlüssel darf keine Teilschlüssel mit gleichem Wert enthalten.
  - Es dürfen nicht mehrere Schlüssel definiert werden, welche denselben Wert haben. Um diese Bedingung zu prüfen, müssen alle Teilschlüssel kombiniert werden.
  - Der Benutzer wird gewarnt, wenn keine Schlüssel erfasst wurden.
- `SecretInputActivity`
  - Das eingegebene Text-Secret darf nur unterstützte Zeichen enthalten.
  - Das eingegebene Secret muss kleiner als der gewählte Secret Space sein.

6.5.11. QR Codes mit ZXing

Wie schon in Kapitel 4.9.12 erwähnt, kann der Barcode-Leser von ZXing [50] auf Android für die Verarbeitung von QR-Codes verwendet werden. Um die Integration des Barcode-Lesers zu vereinfachen, stellen die Entwickler von ZXing [50] die Klasse `IntentIntegrator` zur Verfügung. Diese wird verwendet, um einen `Intent` an die betreffende Activity des Barcode-Lesers zu senden. Die Nutzung des `IntentIntegrator`'s bringt den Vorteil, dass bei fehlender Applikation eine Meldung angezeigt wird und der Benutzer wählen kann, den Barcode-Leser vom Android Market zu installieren. Wir haben die erwähnte Klasse in unsere Applikation inkludiert.

6.5.12. Import / Export

Für die Import und Export Funktionen der Android Applikation werden die Klassen aus der Fassade (siehe Abbildung 6.2) verwendet, um einen Store der Applikation zu exportieren oder zu importieren.

Für den Export wird auch ein QR-Code angeboten. Da die Barcode-Applikation von ZXing nur Text verarbeiten kann, haben wir die binäre und komprimierte JSON Repräsentation eines Stores in Base64 [62] enkodiert. Damit verliert man einen Teil der Grössen-Optimierung, weil Base64 einen Overhead von etwa einem Drittel produziert. Trotzdem sind die Base64 enkodierten Daten kleiner als das reine JSON Format. Für den Import wird umgekehrt zuerst eine Base64 dekodierung vorgenommen, bevor die Daten an die Klassen aus der Fassade übergeben werden.

7. Schlussfolgerungen

KryptonIT Methode Bei der Arbeit mit der KryptonIT Methode wurden zwei konkrete Mängel in der zur Verfügung gestellten Bibliothek gefunden:

- Die Kapselung der Zufallsfunktion innerhalb der Bibliothek ist ungünstig. Besser wäre es, wenn die Zufallsfunktion vom Aufrufenden mitgegeben werden kann. Dies wurde beim Erstellen der graphischen Darstellung in der GWT Applikation festgestellt.

- Mit Hilfe der Visualisierung innerhalb der Web Applikation wurde herausgefunden, dass die in der Bibliothek verwendeten Hash-Funktionen für Zahlen grösser als 2^{128} nicht mehr funktionieren. Das führte zu einer ungleichmässigen Verteilung der Resultate, obwohl sie zufällig über den ganzen Wertebereich hätten verteilt werden sollen. Um dies zu beheben, wurde die modulare Exponentiation als Hash-Funktion hinzugefügt.

Allgemein Obwohl eine gemeinsame Abstraktion in Form der Fassade geschaffen wurde, blieben die Schnittfläche und die Gemeinsamkeiten der Arbeiten gering. Dies zeigt sich vor allem in den vorherigen Kapiteln „Analyse“, „Design“ und „Umsetzung", welche jeweils ein unabhängiges Unterkapitel „Web“ und „Mobile“ beinhalten. Bei einer erneuten Durchführung einer Arbeit dieser Art, würden wir uns für zwei separate Arbeiten entscheiden. Nichtsdestotrotz konnten wir uns gegenseitig unterstützen und ergänzen. Die Vorteile eines „gemeinsamen Projekts“ waren aber kleiner als erwartet.

Zu Beginn der Arbeit erschienen die Ziele der Arbeit als realistisch und der Umfang nicht zu gross. Wie wir schon so oft in Informatik-Projekten selbst erfahren haben, liegt der „Teufel im Detail“. Gegen Ende der Arbeit haben wir festgestellt, dass der geplante Umfang um einiges grösser war, als angenommen. Mit gemeinsamen Anstrengungen gelang es schliesslich trotzdem, alle notwendigen Anforderungen umzusetzen.

Web Applikation - Jan Liechti Alle definierten Muss-Kriterien konnten erfolgreich umgesetzt werden. Es war eine sehr interessante und neue Erfahrung für mich, da ich das erste Mal mit GWT gearbeitet habe und auch zum ersten Mal ein Projekt in dieser Grösse umsetzen durfte. Leider wurde der Aufwand für die Umsetzung der GWT Applikation grundsätzlich unterschätzt. Weil mir die Erfahrung mit GWT fehlte, traten immer wieder kleine Stolpersteine auf, welche die Arbeit verzögerten. Weiter habe ich die Erkenntnis gemacht, dass viele der von mir untersuchten Bibliotheken mit der verwendeten GWT Version nicht funktionieren (im Vergleich zu älteren GWT Versionen). Dies spiegelte sich dadurch wieder, dass die Umsetzung der Applikation im Moment nicht komplett clientseitig möglich ist, was für mich sehr enttäuschend war. Trotzdem wurde immer wieder versucht, die einzelnen Bestandteile der Applikation so umzusetzen, dass ein Umbau auf eine clientseitige Applikation keinen grossen Aufwand darstellen wird. Dieses Vorgehen machte die Umsetzung wiederum etwas schwieriger.

Mobile Applikation - Louis Bernath Das Arbeiten mit Android hat mir grossen Spass gemacht. In der aktuellen Version macht sowohl die Entwicklungs-Umgebung, sowie das Betriebssystem einen sehr ausgereiften und stabilen Eindruck. Interessant war auch, den Horizont in einem anderen Umfeld und mit einer neuen Technologie zu erweitern. Alle Muss-Kriterien und einige Kann-Kriterien konnten für die Mobile Applikation umgesetzt werden. Die Kann-Kriterien im Bereich der zusätzlichen Sensor-Eingaben wurde allerdings unterschätzt. Sogar das Beziehen von Daten aus einem NFC Tag hat sich als weitaus schwieriger erwiesen, als es zu Beginn den Anschein machte. Das Auslesen und Visualisieren von zusätzlichen Sensoren müsste in weiteren Projekten untersucht und umgesetzt werden. Die Applikation wurde so entworfen, dass das Hinzufügen weiterer Eingaben ohne Probleme möglich ist.

Ausblick

Für die „Benutzer“- sowie die „Erklärende“-Sicht wäre es wünschenswert, ein konkretes Anwenderszenario zu definieren, damit sowohl die Methode, wie auch die beiden Applikationen auf ein spezifisches Szenario weiterentwickelt werden könnten. Weiter kommt hinzu, dass die formale Verifikation der Methode noch ausstehend und damit für den täglichen Einsatz noch ungeeignet ist.

Web Applikation Um die Web Applikation als weiterführendes Material für die Lehre einsetzen zu können, wäre es nützlich, wenn neben den berechneten Werten auch noch die durchgeführten Berechnungen angezeigt würden. Weiter müssten noch die Texte für die Hilfe-Anzeigen erstellt und bei den Platzhaltern eingebaut werden. Falls die Web Applikation auch praktisch als Passwort-Manager eingesetzt werden soll, muss die Applikation komplett clientseitig ausgeführt werden können. Hierfür müssten alle serverseitigen Methoden umgeschrieben werden. Um den graphischen Bereich interessanter zu machen, wäre eine Zoomfunktion für das Vergrössern und Verkleinern des SVG-Grafik sehr wirkungsvoll.

Mobile Applikation Als erste Anpassung der Android Applikation würden wir die Reihenfolge der Schlüssel respektive Geheimnis-Eingabe bei der Erstellung vorschlagen. Zuerst sollten die Geheimnisse, dann die Schlüssel eingegeben werden. Mit dieser Änderung könnten Geheimnisse selbst wieder als Schlüssel verwendet werden und damit eine Art Geheimnis-Schlüssel Schleife erstellt werden. Des Weiteren sollten zusätzliche Eingaben untersucht und umgesetzt werden. Dabei muss untersucht werden, welche Sensoren wie verwendet werden könnten und wie diese für den Benutzer visualisiert werden. Um eine Nutzung für End-Benutzer zu erlauben, sollten Texte erstellt und in das vorhandene Hilfesystem einpflegt werden. Der Import und Export per QR-Code könnte weiter optimiert werden, indem der Inhalt binär übertragen würde, sodass grössere Stores ausgetauscht werden könnten.

A. Grundlagen

In diesem Kapitel werden die Grundlagen für die kryptographische Methode erklärt. Mit der Methode kann ein Secret Storage System, also ein System, um Geheimnisse in einem Chiffrat zu speichern, erstellt werden. Falls nicht anders vermerkt, sind die Inhalte aus der Arbeit „Projekt 2“ übernommen worden.

A.1. Secret-Storage System

Das Secret-Storage System hat folgende Eigenschaften:

- Der Benutzer kann n Schlüssel (keys) frei wählen. (tiefe Entropie)
- Der Benutzer kann n Geheimnisse (secrets) frei wählen. (hohe Entropie)
- Der Benutzer kann mehrere Geheimnisse in einem Storage abspeichern.
- Der Benutzer darf nur das Geheimnis erhalten, welches mit dem eingebenen Schlüssel verbunden ist.
- Das System hat alle Eigenschaften eines (symmetrischen) Crytpo-Systems.

Grundprinzip

Das Grundprinzip des Secret-Storage Systems besteht aus den grundlegenden Funktionen „Encrypt Passwords“ und „Decrypt Password“.

Bei der Funktion „Encrypt Passwords“, wird eine Liste von n Passwörtern, also eine Liste von Schlüsseln (keys) $\langle k_1, \ldots, k_n \rangle$ und die zugehörigen Geheimnisse (secrets) $\langle s_1, \ldots, s_n \rangle$, definiert und danach im Store c abgelegt.

Bei der Funktion „Decrypt Password“, wird anhand der Eingabe eines bestimmten Schlüssels k_i das zugehörige Geheimnis s_i abgefragt und zurückgegeben.

$$\mathrm{encrypt}_k(s) = c$$
$$\mathrm{decrypt}_{k_i}(c) = s_i$$

Weitere wichtige Eigenschaften des Secret-Store Systems sind: [2]

- Beim Erhalt eines Geheimnisses s_i werden keine Informationen über die anderen Geheimnisse des Stores c preisgegeben.
- Gibt man den Schlüssel k_i im Store ein, wird das Geheimnis s_i zurückgegeben.
- Durch den Besitz eines Stores ist kein Rückschluss auf die gespeicherten Secrets möglich.
- Bei der Verwendung eines zufälligen k_j ist die Wahrscheinlichkeit, ein s_i aus dem Store c zu erhalten, $\frac{1}{|S|}$.
- Das System ist „non-challenging“ - d.h. der Angreifer (oder auch der Benutzer) kann nicht sagen, ob durch die Verwendung eines Schlüssels k ein zuvor gespeichertes Secret s zurückgegeben wird.

Kombinieren von Schlüsseln

- Oder - Durch das Abbilden von k_i und k_j auf s_i wird dasselbe Geheimnis s_{ij} auf mehrere unabhängige Schlüssel verteilt.
- Und - Durch das Verknüpfen von k_i und k_j zu k_y und anschliessendes Abbilden auf s_y, müssen beide Schlüssel verwendet werden, um das Geheimnis s_y zu erhalten.

Secret Storage System mit Polynomen im endlichen Körper

In diesem Kapitel wird ein Secret-Storage System umrissen, welches die oben geforderten Bedingungen erfüllt. Im Kern basiert die Lösung auf der Möglichkeit, aus n Tupel (k_i/s_i) ein Polynom zu interpolieren. Dieses wird im endlichen Körper $\mathbb{Z}_p$ interpoliert, so ist das Polynom diskretisiert. Es treten daher nur ganzzahlige Koeffizienten für das interpolierte Polynom auf. [2]

Erweitern des Key-Raumes Die verwendeten Keys $k \in \mathbb{Z}$ müssen kollisionsfrei nach $x \in \mathbb{Z}_p$ abgebildet werden können. Dazu wird die Funktion $keyEnhance(k_i+k_0)$ verwendet. Dabei ist die Funktion $keyEnhance$ eine Einwegfunktion, welche die Parameter k_i und k_0 (wird im weiteren Verlauf des Dokuments auch Offset genannt) konkatiniert. Dabei muss der Offset für die gegebenen Keys so gewählt werden, dass die Abbildung kollisionsfrei ist.

Reduzieren des Secret-Raumes Werden die Secrets direkt in der Menge $\mathbb{Z}_p$ abgebildet, so ist die Wahrscheinlichkeit, dass mit einem beliebigen Polynom ein Wert s auf der Secret-Achse zu einem Wert k auf der Key-Achse existiert $\frac{1}{e}$. Dieser Umstand könnte für eine Attacke ausgenutzt werden. Um diese Schwäche aufzuheben, wird ein reduzierter Secret-Raum S definiert. Um von einem Secret $s_i \in S$ zu $y_i \in \mathbb{Z}_p$ abzubilden, können wir die Funktion

$$y_i = s_i + r \times |S|, r \in \mathbb{N}_0 y_i < p, s_i \leq |S|$$

verwenden. Der Parameter r ist dabei eine beliebige Zahl, welche die obigen Bedingungen erfüllt. Und umgekehrt von y_i zu s_i ist die Umkehrfunktion

$$s_i = y_i \, mod \, |S|$$

definiert.

Wahl von p Die Primzahl p für den endlichen Körper $\mathbb{Z}_p$ wird mindestens 2^u mal grösser als der Secret-Raum $|S|$ gewählt. Dabei kann u als den Security Parameter betrachtet werden.

A.2. Polynomial Interpolation

Bei der Polynomial Interpolation suchen wir ein Polynom P, bei welchem der Graph G_p durch alle definierten Stützpunkte $(x_1, y_1), \ldots, (x_n, y_n)$ geht, wobei $x_i, y_i \in \mathbb{R}$. Das gesuchte Polynom besitzt bei n Stützpunkten höchstens den Grad $n - 1$, falls ein Polynom minimalen Grades gesucht wird und die Punkte nicht speziell (z.B. in einer Gerade) liegen. Bei den gegebenen Punkten nennt sich die x-Koordinate „Stützstelle“ und die y-Koordinate „Stützwert“ des Punktes.

Newtonscher Algorithmus

Der Newtonsche Algorithmus wird hier nicht formal definiert, sondern es soll die grundlegende Vorgehensweise des Algorithmus vermittelt werden. Gegeben sind die Stützstellen $(x_1, y_1), \ldots, (x_n, y_n)$. Wir verwenden die Newtonsche Interpolationsformel, um das Polynom zu berechnen.

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0) \cdots (x - x_{n-1})$$

Um $a_0 ... a_n$ zu erhalten, verwenden wir das rekursive Schema der dividierten Differenzen. Dieses wird hier zur einfacheren Verständlichkeit nur Beispielhaft mit vier Stützpunkten aufgezeigt.

$$
\begin{array}{llllll}
x & f(x) & & & & \\
x_1 & y_1 & & & & \\
 & & \frac{y_2-y_1}{x_2-x_1} = b_1 & & & \\
x_2 & y_2 & & \frac{b_2-b_1}{x_3-x_1} = c_1 & & \\
 & & \frac{y_3-y_2}{x_3-x_2} = b_2 & & \frac{c_2-c_1}{x_4-x_1} = d_1 & \\
x_3 & y_3 & & \frac{b_3-b_2}{x_4-x_2} = c_2 & & \\
 & & \frac{y_4-y_3}{x_4-x_3} = b_3 & & & \\
x_4 & y_4 & & & &
\end{array}
$$

Nun kann $a_0, \ldots, a_n$ aus der obersten Diagonale ab y_1 ausgelesen werden, für obiges Beispiel gilt also $a := y_1, b_1, c_1, d_1$. [63]

A.3. Einwegfunktionen

Eine Einwegfunktion in der Informatik ist per Definition eine einfach auszuführende, aber nur schwierig umzukehrende Funktion. Für die Kryptographie sind solche Einwegfunktionen von grosser Bedeutung.

Folgende Eigenschaften werden von Einwegfunktionen für die Kryptographie gefordert, damit diese als sicher gelten: [1] [64]

1. **Urbild Beständigkeit (respektive nicht umkehrbar)**
   Für eine Ausgabe z soll das Berechnen von x für $z = h(x)$ nicht durchführbar sein.

2. **Zweite Urbild Beständigkeit**
   Für die Eingaben x und y, wobei $x \neq y$, so soll die Ausgabe auch unterschiedlich, also $h(x) \neq h(y)$ sein.

3. **Kollisions-Beständigkeit**
   Für zwei beliebige Eingaben x und y, wobei $x \neq y$, soll es unmöglich sein, zwei gleiche Ausgaben, also $h(x) \equiv h(y)$ zu berechnen.

Hashfunktionen

Eine Untergruppe der Einwegfunktionen stellen die Hash-Funktionen dar. Diese werden in der Informatik für Datenbank-Indizes, Kryptologie und das berechnen von Prüfsummen verwendet. Die bekanntesten Vertreter von Hash-Funktionen sind die Algorithmen MD5 und SHA-1 (und auch die Weiterentwicklung SHA-2). Es gibt aber eine Vielzahl anderer Algorithmen und noch eine grössere Menge von Implementationen davon. [65] [64]

[1] In der Vergangenheit wurde Schwächen in kryptographisch verwendeten Einweg-Funktionen ausgenutzt um Kryptographische System zu knacken. Die hier definierten Eigenschaften sind für eine ideale Einwegfunktion.

A.4. Finite Modulare Gruppen

Finite Modulare Gruppen basieren auf der Modulo Arithmetik. Als Grundlage betrachten wir zuerst die für unsere Zwecke relevanten Operationen, die Modulare Addition, die Modulare Multiplikation und die Kongruenz.

Modulare Addition

$$a \oplus b := ((a \bmod n) + (b \bmod n)) \bmod n = (a + b) \bmod n$$

Modulare Multiplikation

$$a \odot b := ((a \bmod n) \times (b \bmod n) = (a \times b) \bmod n$$

Kongruenz – Modulare Gleichheit

$$(a \equiv b \bmod n) \quad \leftrightarrow \quad (a \bmod n = b \bmod n)$$

Die modulare Gruppe $\mathbb{Z}_n$ ist als endliche Menge $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ definiert, mit den entsprechenden oben definierten Operationen. Zusätzlich gelten folgende Eigenschaften, welche wir uns zu Nutze machen wollen.

Additiv inverse Elemente

Zu jedem $a \in \mathbb{Z}_n$ gibt es ein inverses Element $\bar{a} \in \mathbb{Z}_n$ so dass $a \oplus \bar{a} = \bar{a} \oplus a = 0$. Durch diese Eigenschaft können wir eine Subtraktion (in einer gegebenen modularen Gruppe) mittels Addition definieren. So interpretiert man $a - b = c$ mittels additiv inversem Element als $a \oplus \bar{b} = c$.

Multiplikativ inverse Elemente

Für jedes $a \in \mathbb{Z}_n \backslash \{0\}$ gibt es nur dann ein inverses Element $a^{-1} \in \mathbb{Z}_n$, wenn n prim ist, so dass $a \odot a^{-1} = 1$. Wie auch schon bei der Addition können wir uns hier das inverse Element zu Nutze machen, indem wir $\frac{a}{b} = c$ als $a \odot b^{-1} = c$ interpetieren.

Prime Gruppen

Um die multiplikativ-inversen Elemente für a in $\mathbb{Z}_n$ zu finden, verwenden wir den erweiterten euklidischen Algorithmus. Da wir für jedes a in $\mathbb{Z}_n$ ein multiplikativ inverses Element benötigen, verwenden wir ausschliesslich finite Gruppen $\mathbb{Z}_n$, wo n prim ist. Deshalb verwenden wir im weiteren Verlauf dieses Dokuments die Bezeichnung $\mathbb{Z}_p$ um auszudürcken, dass es sich bei p um eine Primzahl und bei $\mathbb{Z}_p$ um eine prime Gruppe handelt.

B. Analysen

B.1. Text Encoding und die Grösse des Secret Space mit Text Geheimnissen

Wollen wir ein 8-stelliges Passwort im ASCII (7 Bit) Zeichenraum abbilden, benötigen wir einen Secret Space von mindestens $2^{(7\times8)} \approx 7.205 \times 10^{13}$. Möchte man ein modernes Encoding wie UTF-8 verwenden, werden zwischen 8 und 32 Bit pro Symbol verwendet. So dass ohne Kenntnis der verwendeten Symbole mindestens mit 32 Bit pro Zeichen gerechnet werden müsste. Bei UTF-16 oder UFT-32, werden immer 16 respektive 32 Bit pro Zeichen verwendet. Das würde bei einem 8 Zeichen langen Passwort einen Secret Space von mindestens $2^{32\times8} \approx 3.403 \times 10^{38}$ respektive $2^{32\times8} \approx 1.157 \times 10^{77}$ benötigen. Die Grösse der erwähnten Secret Spaces sind grundsätzlich kein Problem für die KryptonIT Methode. Das Problem ergibt sich durch in den niedrigen Anteil des benutzten Bereiches, da eine grosse Anzahl von mit hoher Wahrscheinlichkeit ungenutzten Symbolen existiert.

Um dies zu umgehen, aber trotzdem eine gewisse Anzahl an möglichen Symbolen anzubieten, wird das Encoding auf eine bei Passwörter üblichen Symbolen beschränkt.

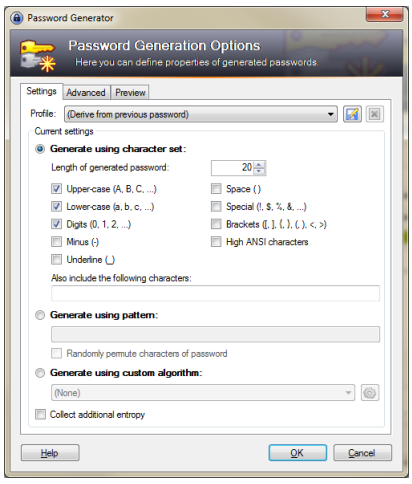


Abbildung B.1.: KeePass Passwort Generator

- [A - Z] Grossbuchstaben
- [a - z] Kleinbuchstaben
- [0 - 9] Zahlen
- [ ] Leerzeichen
- [! "# $ % & ' * + , - . / : ; = ? \ ^ _ | ~ @] Sonderzeichen
- [() [] <>] Klammern

So können 82 verschiedene Zeichen für die Eingabe des Geheimnisses verwendet werden. Für die Umrechnung aus der definierten Menge von Zeichen in eine Zahl und wieder Zurück wird die Fassade erweitert. Die Umrechnung muss vom dezimalen System in das Basis-82 System und umgekehrt erfolgen. Mit Hilfe einer Symbol-Tabelle kann das Zeichen dann in der Zahl Basis-82 abgelesen werden. Die Symbol-Tabelle nummeriert die Zeichen bei 0 beginnend bis 81 durch ($[0 = A, 1 = B, \ldots, <= 80, >= 81]$).

B.2. Store Formate

Hier ein Beispiel für einen Store in JSON:

```
{"secretSpace":"10","prime":"13","name":"test","order":"16","polynom":["{"
    exponent":"0","coeffizient":"55"}","{"exponent":"1","coefficient":"79"}"],
    "kappa":53","hash":"SHA1"}
```

Derselbe Store in einer XML Stuktur:

```
<store secretSpace="10" prime="13" name="test" order="16" kappa="53" hash="
    SHA1">
        <polynom>
                <monomial exponent="0" coeffizient="55"/>
                <monomial exponent="1" coefficient="79"/>
        </polynom>
</store>
```

B.3. Web Applikation

B.3.1. KryptonIT Bibliothek mit Google Web Toolkit

Folgende Tests und Analysen wurden durchgeführt um sicherzustellen, dass die KryptonIT Bibliothek eingebunden werden kann und korrekt funktioniert, da sie den Kern der Anwendung darstellt.

Download und Einfügen der KryptonIT Bibliothek als Modul in ein neues GWT Projekt

Damit die KryptonIT Bibliothek komplett clientseitig funktioniert, muss diese ins Projekt eingebunden werden, damit Sie beim Kompilieren in JavaScript übersetzt werden kann. Eingebunden wird die Bibliothek als Modul, wofür eine gwt.xml Datei erstellt wird, in welcher alle benötigten Packages definiert werden. Das Kompilieren des Projekts endete fehlerhaft. Es werden zusätzliche GWT Bibliotheken benötigt, um die KryptonIT Bibliothek so kompilieren zu können, dass diese in JavaScript übersetzt werden kann.

```
Compiling module ch.bfh.kryptonIT.KryptonIt
   Validating newly compiled units
      Ignored 6 units with compilation errors in first pass.
Compile with -strict or with -logLevel set to TRACE or DEBUG to see all errors.
   [ERROR] Errors in 'file:/home/bangster/workspace/KryptonIt/src/ch/simplix/util/hash/implementation/HashMac.java'
      [ERROR] Line 7:  The import javax.crypto cannot be resolved
      [ERROR] Line 8:  The import javax.crypto cannot be resolved
      [ERROR] Line 57:  Mac cannot be resolved to a type
      [ERROR] Line 57:  Mac cannot be resolved
      [ERROR] Line 58:  SecretKeySpec cannot be resolved to a type
      [ERROR] Line 58:  SecretKeySpec cannot be resolved to a type
   [ERROR] Cannot proceed due to previous errors
```

Abbildung B.2.: Fehler beim Versuch die KryptonIT Library in JavaScript zu kompilieren

Aus der Fehlermeldung im Bild wird ersichtlich, dass die Klassen Mac und SecretKeySpec in den GWT Bibliotheken fehlen. Ob noch mehrere Klassen fehlen, ist aus der Fehlermeldung nicht ersichtlich, da die Kompilierung erst weiterfahren kann, wenn diese Konflikte behoben sind.
Nach kurzer Recherche im Internet und einem Tipp von unserem Betreuer Reto König fanden wir heraus, dass die fehlenden Pakete in der GWT Bibliothek crypto-gwt (Version 1.0) vorhanden sind. [66]

Download und Einfügen der fehlenden Bibliothek (crypto-gwt) als Modul ins Projekt

Das Kompilieren des Projekts endete wieder fehlerhaft. Nun fehlten Dateien einer Bibliothek (com.googlecode.future.AsyncFuture), welche in der crypto-gwt Bibliothek benötigt wurden. Die fehlenden Dateien konnten nach weiteren Recherchen im Internet gefunden werden.[67]

Leider waren die crypto-gwt Bibliothek und die gwt-async-future Bibliothek zueinander nicht 100% kompatibel (Versionsunterschiede). Somit kamen wir so nicht weiter.

Download und Einfügen der fehlenden Bibliothek (crypto-gwt) ins Projekt mit Maven

Da das Projekt auch über Maven verfügbar war, versuchten wir auch diesen Weg. Maven erkannte sofort, dass die Bibliothek gwt-async-future fehlte, hat aber den Konflikt auflösen können und die benötigte Bibliothek gefunden und auch heruntergeladen.
Zudem konnte das Projekt nun auch in JavaScript kompiliert werden. Nach einigen Funktionstests, die Applikation clientseitig im Browser laufen zu lassen, stellte sich aber heraus, dass das automatische Ersetzen

der Java Klassen durch die eingebundenen GWT Klassen crypto-gwt nicht funktionierte und somit die benötigten Klassen aus der crypto-gwt Bibliothek nicht korrekt in JavaScript übersetzt wurden. Das Ergebnis in der Applikation war, dass diese einfach nichts machte, anstatt der erwarteten Aktionen. Die Anwendung blieb hängen.

Beim näheren Betrachten des Maven-Manifests von der crypto-gwt Bibliothek zeigte sich, dass diese nur GWT 1.7.1 unterstützt. Wir verwenden die aktuelle Version GWT 2.4.0. und App Engine 1.5.2.

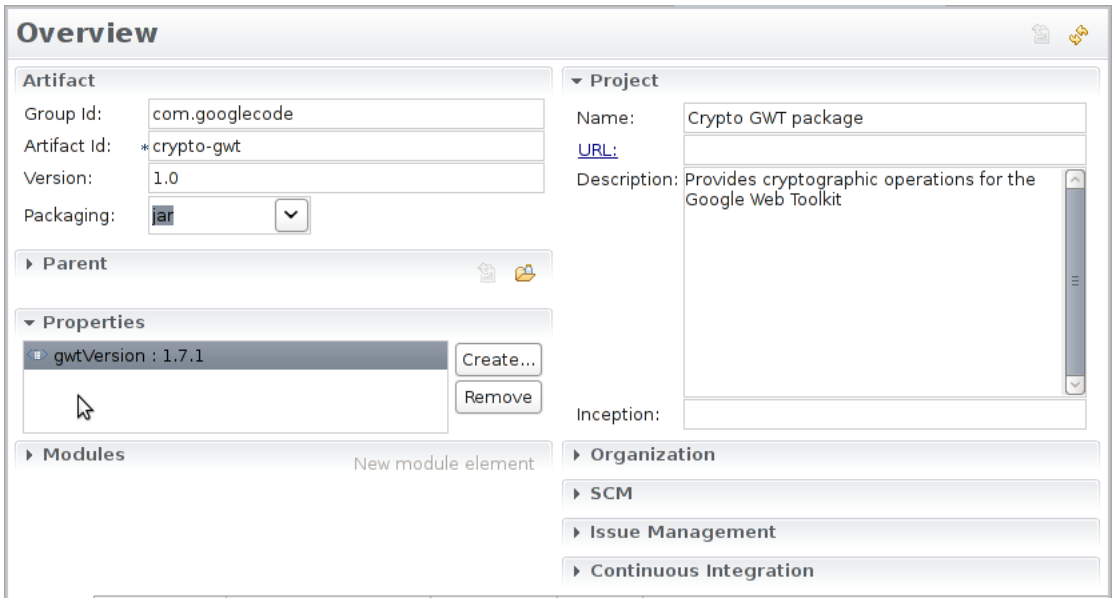


Abbildung B.3.: Manifest Datei der crypto-gwt Bibliothek

Wahrscheinlich müsste man hier nur noch ein bisschen warten, bis die benötigten Bibliotheken für die neue GWT Version erstellt wären, und dann könnte es funktionieren, die Applikation komplett clientseitig laufen zu lassen.

Die crypto-gwt Bibliothek selbst auf GWT 2.4 anzupassen würde den Rahmen dieser Arbeit sprengen, was also nicht in Betracht gezogen werden kann.

GWT 1.7.1

GWT 1.7.1 kommt nicht in Frage, da diese nicht mehr aktuell ist, und dort die Klasse BigInteger, welche in der KryptonIT Bibliothek benötigt wird, nicht vorhanden ist. Weiter konnte nach kurzer Recherche auch keine ältere App Engine gefunden werden, da GWT 1.7.1 nach einem kurzen Versuch nicht mit der App Engine 1.5.2 funktionierte.

Bibliothek gwt-crypto

Weitere Versuche wurden mit der Bibliothek gwt-crypto[68] durchgeführt. Leider fehlen in dieser Bibliothek aber z.B. die Klassen Mac und SecretKeySpec, welche für die KryptonIT Bibliothek zwingend benötigt werden.

B.3.2. Zeichnen mit GWT

Es wurden folgende unterschiedliche Aktionen durchgeführt um zu Testen, wie mit Google Web Toolkit gezeichnet werden kann.

Als Erstes wurde eine kurze Analyse von der Verwendung von Canvas in GWT mit ein paar Beispielen aus dem Internet gemacht, welche wir von Reto König erhalten haben. Das Canvas-Element ist Bestandteil von HTML5 und gestattet ein dynamisches Rendern von Bitmap-Grafiken.[22]

Da nicht gleich alles auf Anhieb funktionierte, wurde beschlossen, weitere Recherchen durchzuführen.

Eine weitere Besprechung mit Reto König ergab, dass es sowieso besser wäre mit SVG-Grafiken zu arbeiten, da diese Vektor basiert sind.

Scalable Vector Graphics (SVG), (deutsch: skalierbare Vektorgrafik) ist die vom World Wide Web Consortium (W3C) empfohlene Spezifikation zur Beschreibung zweidimensionaler Vektorgrafiken. SVG basiert auf XML.[69]

Also wurde nach Möglichkeiten gesucht, SVG-Grafiken mit GWT zu erstellen.

Als Erstes wurde die Bibliothek lib-gwt-svg von Vectomatic gefunden.[23][24] Nach kurzer Analyse zeigte sich, dass die erforderlichen Komponenten (Linie, Kreis, Text) mit dieser Bibliothek erstellt werden können. Es wurde ein erster simpler Prototyp für die Darstellung eines Koordinatensystems mit dieser Bibliothek erstellt. Das Ergebnis sieht folgendermassen aus.

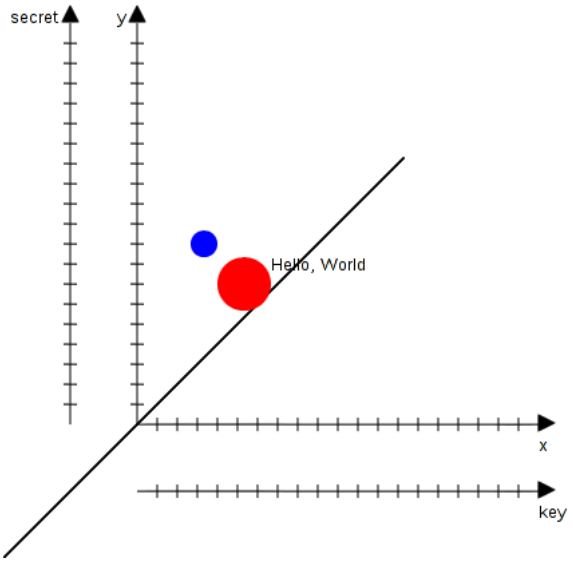


Abbildung B.4.: Prototyp eines Koordinatensystems mit der lib-gwt-svg Bibliothek

Da wir uns nicht mit der erstbesten Lösung zufrieden geben wollten, wurde beschlossen nach weiteren Bibliotheken zu suchen, welche dieselbe Funktionalität bieten. Nach einer weiteren Recherche im Internet wurde die Bibliothek gwt-graphics auf der Google Code Webseite gefunden.[25][26] Diese Bibliothek bietet die gleiche Funktionalität wie die Bibliothek lib-gwt-svg.

Um die beiden Bibliotheken miteinander zu Vergleichen, wurde auch mit der zweiten Bibliothek ein identischer Prototyp erstellt.

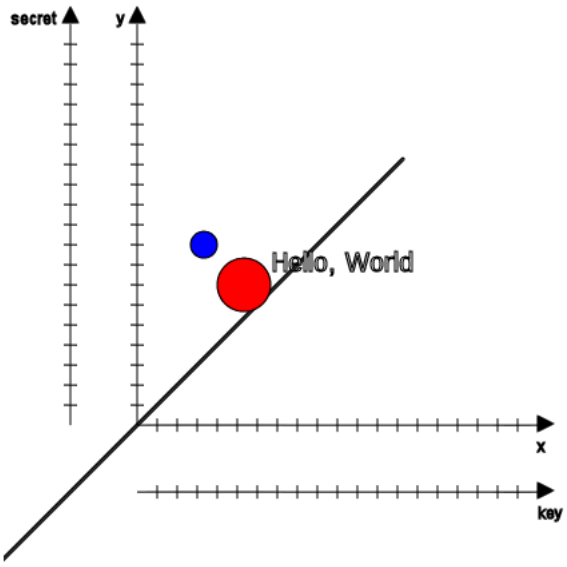


Abbildung B.5.: Prototyp eines Koordinatensystems mit der gwt-graphics Bibliothek

Das Ergebnis der grafischen Ausgabe ist bei beiden Prototypen fast identisch und auch der generierte Code ist derselbe, was noch nicht ausreichte, um sich für eine der Bibliotheken zu entscheiden.

Eine Betrachtung des geschriebenen Codes führte aber klar zur Entscheidung.

```
OMSVGEllipseElement ellipse = doc.createSVGEllipseElement(150, 180, 10, 10);
ellipse.getStyle().setSVGProperty(SVGConstants.CSS_FILL_PROPERTY, "blue");

OMSVGLineElement xLine = doc.createSVGLineElement(100, 315, 400, 315);
xLine.getStyle().setSVGProperty(SVGConstants.CSS_STROKE_VALUE, SVGConstants.CSS_BLACK_VALUE);
xLine.getStyle().setSVGProperty(SVGConstants.SVG_STROKE_WIDTH_VALUE, "1");

OMSVGLineElement keyLine = doc.createSVGLineElement(100, 365, 400, 365);
keyLine.getStyle().setSVGProperty(SVGConstants.CSS_STROKE_VALUE, SVGConstants.CSS_BLACK_VALUE);
```

Abbildung B.6.: Codeausschnitt bei Verwendung der lib-gwt-svg Bibliothek

```
Circle ellipse = new Circle(150, 180, 10);
ellipse.setFillColor("blue");

Line xLine = new Line(100, 315, 400, 315);
xLine.setStrokeColor("black");
xLine.setStrokeWidth(1);

Line keyLine = new Line(100, 365, 400, 365);
keyLine.setStrokeColor("black");
```

Abbildung B.7.: Codeausschnitt bei Verwendung der gwt-graphics Bibliothek

Mit der gwt-graphics Bibliothek sind die Klassennamen intuitiver und kürzer benannt, was den Code eleganter und viel besser lesbar macht. Zudem wurde das Handling mit dieser Bibliothek ebenfalls als einfacher und intuitiver empfunden.

Anhand dieser Erkenntnisse wurde der Entscheid getroffen, die Bibliothek gwt-graphics in diesem Projekt einzusetzen.

B.3.3. Sicherheit des Systems

Um die Webapplikation sicher machen zu können und sie dann zusätzlich auch in der eigentlich dafür gedachten Funktion als Secret Store Manager nutzen zu können, müssten die bereits erwähnten Probleme mit den fehlenden GWT Bibliotheken gelöst sein. Wenn alle Benötigen GWT Bibliotheken vorhanden sind, sollte mit geringem Anpassungsaufwand die Applikation so umgeschrieben werden können, dass sie komplett clientseitig im Browser des Benutzers läuft und somit keine sensiblen Daten mehr übers Netz schickt, da keine Client-Server Kommunikation mehr stattfindet.

Eine weitere Möglichkeit wäre, die KryptonIT nur soweit wie möglich clientseitig zu implementieren und nur die Teile auf dem Server auszuführen oder zu berechnen, welche clientseitig nicht möglich sind. Dies würde aber nur aufzeigen, dass grundsätzlich die Möglichkeit bestehen würde, alles clientseitig zu implementieren. Trotzdem wäre es nur ein Trugbild zu denken, die Applikation wäre so sicherer. Diese Möglichkeit wäre also nur eine weitere Umgehungslösung und würde einem zwar einen Schritt weiter, aber trotzdem noch nicht zu einer sicheren Applikation führen. Diese Möglichkeit wird in Betracht gezogen, wenn noch genügend Zeit vorhanden ist.

B.4. Mobile Applikation

B.4.1. Himmelsrichtungen

Himmelsrichtung	Schlüssel-Wert	Bereich
Norden	N	$337.5°$ - $0°$ oder $0°$ - $22.5°$
Nordosten	NO	$22.5°$ - $67.5°$
Osten	O	$67.5°$ - $112.5°$
Südosten	SO	$112.5°$ - $157.5°$
Süden	S	$157.5°$ - $202.5°$
Südwesten	SW	$202.5°$ - $247.5°$
Westen	W	$247.5°$ - $292.5°$
Nordwesten	NW	$292.5°$ - $337.5°$

C. Implementation

C.1. Store Sicherheits Stufen

Die drei Sicherheitsstufen und deren Werte. Die „Sicherheitsparameter 1 & 2“ richten sich nach den Empfehlungen vom Autor der KryptonIT Bibliothek. Die Werte für den Parameter „Zufallspunkte“ repräsentieren ein (abgeschlossenes) Intervall, in welchem die Anzahl der Punkte ausgewählt werden.

	Normal	Hoch	Verrückt
Sicherheitsparameter 1 (Grad)	2^{100}	2^{500}	2^{1000}
Sicherheitsparameter 2 (Faktor)	2^{100}	2^{500}	2^{1000}
Zufallspunkte (min-max)	$3-5$	$7-10$	$15-20$
Hash-Methode	EXP	EXP	EXP

Mit der Abkürzung „EXP“ ist die modulare Exponentiation gemeint. Näheres dazu kann in Kapitel 6.3 gefunden werden.

C.2. Web Applikation

C.2.1. Entwicklungsumgebung

Für die Entwicklung der GWT Applikation wurden folgende Softwareversionen verwendet:

- Eclipse Java EE IDE for Web Developers (Helios Service Release 2)
- App Engine SDK (App Engine - 1.5.2)
- GWT SDK (GWT - 2.4.0.rc1)
- GWT Graphics Library (gwt-graphics 1.0.0)
- Browser (Iceweasel 5.0, Chromium 13.0.782.107, Firefox 7.0.1)

C.2.2. KryptonIT-Service

Folgende Methoden wurden im KryptonIT-Service erstellt:

- getSpace(int numberOfDigits, int securityOrder, int spaceFactor, SecretType secretType) : Erstellt den

Finiten Körper und gibt ihn als SpaceCreateData Objekt zurück. Wird beim Erstellen eines Stores verwendet.

- getKappa(SpaceData space, List<BigInteger> keys, String hashMethod) : Erstellt das KappaData Objekt mit den Schlüsseln und gibt es zurück. Wird beim Erstellen eines Stores verwendet.
- getSigmaToY(SpaceData space, List<BigInteger> secrets) : Erstellt die Geheimnis und gibt sie als List<BigInteger> Objekt zurück. Wird beim Erstellen eines Stores verwendet.
- getStore(String name, SpaceCreateData space, BigInteger kappaOffset, String hashMethod, List<BigInteger> xList, List<BigInteger> yList) : Erstellt ein Chiffrat und gibt es als SecretStoreData Objekt zurück. Wird beim Erstellen eines Stores verwendet.
- getKappa2(SpaceData space, List<BigInteger> keys, String hashMethod, BigInteger kappaOffset) : Erstellt das KappaData Objekt mit den Schlüsseln und gibt es zurück. Wird beim Abfragen eines Stores verwendet.
- getY(SecretStoreData store, List<BigInteger> xList) : Berechnet die y-Werte anhand der eingegeben x-Werte der Schlüssel und gibt diese als List<BigInteger> Objekt zurück. Wird beim Abfragen eines Stores verwendet.
- getSigmaToSecret(SpaceData space, List<BigInteger> yList) : Berechnet die Geheimnis-Werte anhand der y-Werte und gibt diese als List<BigInteger> Objekt zurück. Wird beim Abfragen eines Stores verwendet.
- getStore(String jsonString) : Erstellt einen Store aus einem JSON-String und gibt diesen als SecretStoreData Objekt zurück. Wird beim Importieren eines Stores verwendet.
- getJsonString(SecretStoreData store) : Erstellt einen JSON-String aus einem Store und gibt diesen als String Objekt zurück. Wird beim Exportieren eines Stores verwendet.
- getCompressedJsonString(SecretStoreData store) : Erstellt einen komprimierten String aus einem Store und gibt diesen als String Objekt zurück. Wird beim Exportieren eines Stores als QR-Code verwendet.
- getConvertedSecretList(List<String> secretListAsString) : Berechnet die Geheimnis-Werte anhand der eingegebenen Geheimnisse und gibt diese als List<BigInteger> Objekt zurück. Wird beim Erstellen eines Stores verwendet.
- getDeconvertedSecret(BigInteger secretAsBigInt) : Berechnet das Geheimnis anhand des Geheimnis-Werts und gibt diesen als String Objekt zurück. Wird beim Abfragen eines Stores verwendet.
- getCombinedKey(List<BigInteger> keys) : Berechnet einen Schlüssel aus den Teilschlüssel-Werten und gibt diesen als BigInteger Objekt zurück. Wird beim Erstellen eines Stores verwendet.
- isValidTextSecret(List<String> secrets) : Überprüft, ob die eingegebenen Geheimnisse keine unerlaubten Zeichen beinhalten und gibt „True“ oder „False“ als Boolean Objekt zurück.

C.2.3. Presenter

Die folgenden zehn Presenter-Klassen wurden erstellt:

- KryptonITPresenter : Presenter für die Main-View „KryptonITView“. Der Main-Presenter der Applikation, welcher alle weiteren Presenter der Applikation erstellt. Steuert die Anzeige des Zeichnungsbereichs und des Informationsbereichs.
- WelcomePresenter : Presenter für die WelcomeView.
- FiniteBodyPresenter : Presenter für das FiniteBodyWidget. Stellt die benötigten Methoden zur Verfügung, um einen Finiten Körper (SpaceCreataData Objekt) zu erstellen.
- KeysPresenter : Presenter für das KeysWidget. Stellt die benötigten Methoden zur Verfügung, um ein KappaData Objekt zusammen mit den Schlüsseln zu erstellen.
- SecretsPresenter : Presenter für das SecretsWidget. Stellt die benötigten Methoden zur Verfügung, um die Geheimnisse zu erstellen.
- PolynomInterpolationPresenter : Presenter für das PolynomInterpolationWidget. Stellt die benötigten Methoden zur Verfügung, um Zufallspunkte und einen Store durch Polynominterpolation zu erstellen.
- SaveStorePresenter : Presenter für das SaveStoreWidget. Stellt die benötigten Methoden zur Verfügung, um einen Store im Browser zu speichern.
- LoadStorePresenter : Presenter für das LoadStoreWidget. Stellt die benötigten Methoden zur Verfügung, um einen gespeicherten oder erstellten Store zu laden.
- RetrieveSecretPresenter : Presenter für das RetrieveSecretWidget. Stellt die benötigten Methoden zur Verfügung, um ein Geheimnis aus einem Store abzufragen.
- ImportExportPresenter : Presenter für die ImportExportView. Stellt die benötigten Methoden zur Verfügung, um einen Store zu importieren, exportieren oder löschen.

C.2.4. Eigene Widgets

Folgende Widgets wurden erstellt:

- FiniteBodyWidget : Beinhaltet alle Auswahl-Felder, welche für die Erstellung des Finiten Körpers benötigt werden.
- KeysWidget : Beinhaltet alle Eingabe-Felder und SubKeysWidgets, welche für die Erstellung des Kappa Objekts mit den Schlüsseln benötigt werden.
- SubKeysWidget : Beinhaltet alle Eingabe-Felder für die Eingabe der Teilschlüssel eines Schlüssels.
- SecretsWidget : Beinhaltet alle Eingabe-Felder, welche für die Erstellung der Geheimnisse benötigt werden.

- PolynomInterpolationWidget : Beinhaltet alle Eingabe-Felder, welche für die Erstellung der Zufallspunkte und die Durchführung der Polynominterpolation (welche den Store erstellt) benötigt werden.
- SaveStoreWidget : Beinhaltet alle Eingabe-Felder, welche für das Speichern eines Stores benötigt werden.
- LoadStoreWidget : Beinhaltet alle Auswahl-Felder und ein Ausgabe-Feld, welche für das Laden eines Stores benötigt werden.
- RetrieveSecretWidget : Beinhaltet alle Eingabe-Felder und ein Ausgabe-Feld, welche für das Abfragen eines Geheimnis aus einem Store benötigt werden.
- ImportWidget : Beinhaltet alle Eingabe-Felder, welche für das Importieren eines Stores benötigt werden.
- ExportWidget : Beinhaltet alle Auswahl-Felder und ein Ausgabe-Feld, welche für das Exportieren eines Stores benötigt werden.
- DeleteWidget : Beinhaltet alle Auswahl-Felder, welche für das Löschen eines Stores benötigt werden.

C.2.5. Events

Folgende Event-Klassen wurden erstellt:

- FiniteBodyCreatedEvent : Wird vom FiniteBodyPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass der Finite Körper berechnet wurde.
- KeysCreatedEvent : Wird vom KeysPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass die Schlüssel-Werte beim Erstellen eines Stores berechnet wurden.
- KappaCreatedEvent : Wird vom KeysPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass das KappaData Objekt mit den x-Werten der Schlüssel beim Erstellen eines Stores erzeugt wurde.
- SecretsCreatedEvent : Wird vom SecretsPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass die Geheimnis-Werte erstellt wurden.
- SigmaCreatedEvent : Wird vom SecretsPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass die y-Werte der Geheimnisse erstellte wurden.
- AddPointsClickedEvent : Wird vom SecretsPresenter abgefeuert un den KryptonITPresenter zu informieren, dass die Punkte aus den x- und y-Werten erstellt wurden.
- AddRandomPointsClickedEvent : Wird vom PolynomInterpolationPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass die Zufallspunkte erstellt wurden.
- StoreCreatedEvent : Wird vom PolynomInterpolationPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass die Polynominterpolation durchgeführt und ein Store erstellt wurde.

- StoreSavedEvent : Wird vom SaveStorePresenter oder vom ImportExportPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass der Store im Browser gespeichert wurde.
- ClearCreateStoreValuesClickedEvent : Wird vom SaveStorePresenter abgefeuert, um den KryptonIT-Presenter zu informieren, dass die graphische Ausgabe und der Informationsbereich auf der Store Erstellen Seite gelöscht werden sollen.
- LoadStoreClickedEvent : Wird vom LoadStorePresenter abgefeuert, um den KryptonITPresenter zu informieren, dass ein Store geladen wurde.
- Keys2CreatedEvent : Wird vom RetrieveSecretPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass der Schlüssel-Wert beim Abfragen eines Stores berechnet wurde.
- Kappa2CreatedEvent : Wird vom RetrieveSecretPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass das KappaData Objekt mit dem x-Wert des Schlüssels beim Abfragen eines Stores erzeugt wurde.
- GetYReceivedEvent : Wird vom RetrieveSecretPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass der y-Wert des Geheimnis berechnet wurde.
- GetSecretReceivedEvent : Wird vom RetrieveSecretPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass der Geheimnis-Wert des Geheimnis berechnet wurde.
- ClearRetrieveStoreValuesClickedEvent : Wird vom RetrieveSecretPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass die graphische Ausgabe und der Informationsbereich auf der Store Abfragen Seite gelöscht werden sollen.
- StoreDeletedEvent : Wird vom RetrieveSecretPresenter abgefeuert, um den KryptonITPresenter zu informieren, dass ein Store gelöscht wurde.
- ErrorEvent : Kann von allen Presentern abgefeuert werden, um den KryptonITPresenter zu informieren, dass ein Fehler aufgetreten ist.

C.2.6. Validierungen

Die folgenden Validierungen wurden erstellt:

Validierungen beim Erstellen eines Stores

- Validierungen beim Schritt: Finiter Körper
  - Die Art der Geheimnisse muss gewählt werden.
- Validierungen beim Schritt: Schlüsseleingabe
  - Zuerst muss der finite Körper erstellt werden.
  - Es dürfen keine leeren Teilschlüssel eingegeben werden.

  - Es dürfen keine identischen Teilschlüssel pro Schlüssel eingegeben werden.

- Validierungen beim Schritt: Geheimniseingabe
  - Zuerst muss der finite Körper erstellt werden.
  - Zuerst muss mindestens ein Schlüssel erstellt werden.
  - Es dürfen keine leeren Geheimnisse eingegeben werden.
  - Wurde „Zahlen“ als Art der Geheimnisse im finiten Körper definiert, dürfen nur Zahlen als Geheimnisse eingegeben werden.
  - Wurde „Text“ als Art der Geheimnisse im finiten Körper definiert, dürfen nur erlaubte Zeichen als Geheimnisse eingegeben werden.
  - Geheimnisse dürfen nicht länger sein, als im finiten Körper definiert wurde.
- Validierungen beim Schritt: Zufallspunkte erstellen
  - Zuerst muss der finite Körper erstellt werden.
  - Zuerst muss mindestens ein Schlüssel erstellt werden.
  - Zuerst muss mindestens ein Geheimnis erstellt werden.
  - Das Eingabefeld „Anzahl der Zufallspunkte“ darf nicht leer sein.
  - Im Eingabefeld „Anzahl der Zufallspunkte“ muss eine Zahl eingegeben werden.
- Validierungen beim Schritt: Polynominterpolation
  - Zuerst muss der finite Körper erstellt werden.
  - Zuerst muss mindestens ein Schlüssel erstellt werden.
  - Zuerst muss mindestens ein Geheimnis erstellt werden.
  - Zuerst müssen die Zufallspunkte erstellt werden.
- Validierungen beim Schritt: Store speichern
  - Zuerst muss ein Store erstellt werden.
  - Es muss ein Name für den Store eingegeben werden.
  - Für den Namen eines Stores dürfen nur erlaubte Zeichen eingegeben werden.

Validierungen beim Abfragen eines Stores

- Validierungen beim Schritt: Store laden

  - Wenn kein temporärer Store geladen ist, muss ein Store ausgewählt werden.

- Validierungen beim Schritt: Store abfragen
  - Ein Store zum Abfragen muss geladen sein.
  - Es dürfen keine leeren Teilschlüssel eingegeben werden.
  - Es dürfen keine identischen Teilschlüssel eingegeben werden.

Validierungen beim Importieren eines Stores

- Die Eingabe des zu importierenden Stores darf nicht leer sein.
- Die Eingabe des zu importierenden Stores müss ein gültiges Format haben.

Validierungen beim Exportieren eines Stores

- Ein Store zum exportieren muss ausgewählt sein.
- Der Export Typ muss ausgewählt sein.

Validierungen beim Löschen eines Stores

- Ein Store zum Löschen muss ausgewählt sein.

C.3. Mobile Applikation

C.3.1. Entwicklungsumgebung

Für die Entwicklung wurden folgende Versionen verwendet:

- Windows 7 x64
- Eclipse Helios Version 3.6.2
- Android Development Tools Version 14 (Eclipse Plugin)
- Android SDK Tools rev. 14
- Android SDK Platform-tools rev. 9
- ObjectAid Class Diagram 1.0.6

Zum Testen der Applikation wurden folgende Android Versionen verwendet:

- Samsung Galaxy S mit Android 2.3.4 (API Level 9)
- HTC Desire S mit Android 2.3.4 (API Level 9)
- Samsung Nexus S mit Android 4.0.3 (API Level 14)
- Android Emulator mit 2.2 (API Level 8)

C.3.2. SQLite

- NULL. The value is a NULL value.
- INTEGER. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- REAL. The value is a floating point value, stored as an 8-byte IEEE floating point number.
- TEXT. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
- BLOB. The value is a blob of data, stored exactly as it was input.

[1]

[1] http://www.sqlite.org/datatype3.html

C.3.3. Klassendiagramm

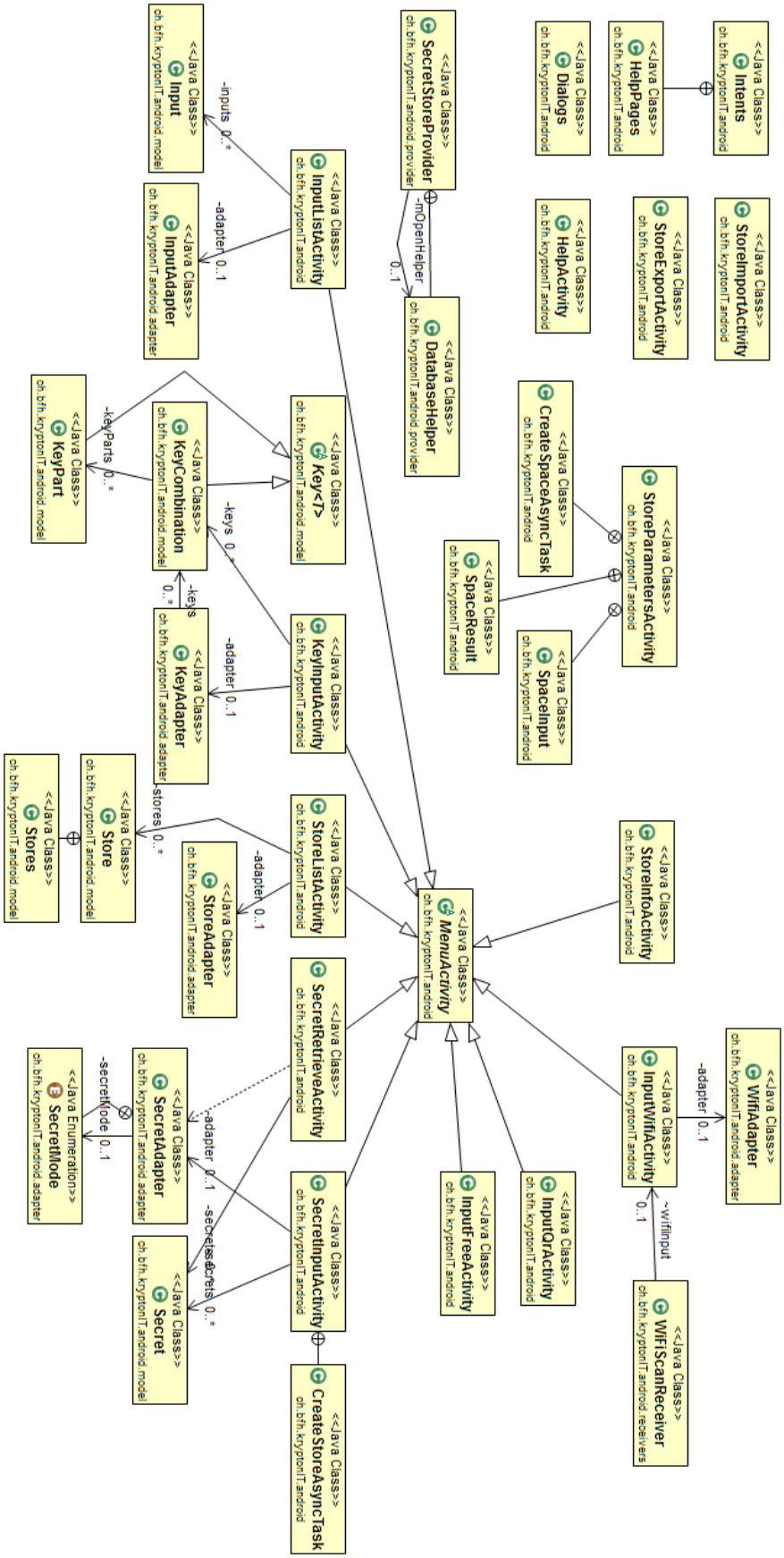


Abbildung C.1.: Android Applikation - Klassendiagramm.

D. Iterationen

D.1. Erste Iteration

D.1.1. Web Applikation

Layout/Benutzeroberfläche

Eine erste Besprechung des Layouts zusammen mit Reto König lieferte folgende Ergebnisse: Die einzelnen Ausgabefelder bei den Knöpfen beinhalten nur zusätzliche Informationen, welche für die Erklärung der Funktionsweise des Passwortmanagers dienen (Erklärende Sicht) und nicht für die eigentliche Erstellung oder das Abfragen eines Stores. Diese Ausgabefelder könnten den Benutzer nur unnötig verwirren. Deshalb sollen diese auf der linken Seite bei den Eingabefeldern und Knöpfen ausgebaut werden und gruppiert in einen separaten Bereich auf der Seite verschoben werden.

Zusätzlich sind bei der Abfrage eines Stores zu viele einzelne Schritte ohne Eingabe (nur nächster Knopf klicken) vorhanden, was es für den Benutzer mühsam und träge macht, einen Store abzufragen (zu viele Klicks). Deshalb sollen bei der Abfrage die Aufteilungsschritte reduziert werden.

Für den Benutzer ist es schwierig, die Bedeutung der einzelnen Eingaben, vor allem beim Erstellen eines Stores, zu verstehen. Deshalb sollen die Eingaben Secret Space und Security Order mit Eingabehilfen versehen werden. Weiter soll der Benutzer bei jeder Eingabe die Möglichkeit haben, sich über die Bedeutung eines Eingabefeldes genauer zu Informieren.

Aufbau der Applikation

Ein bedachter Aufbau der Applikation zu Beginn der Entwicklung ist sinnvoll. Leider wurde bisher zu wenig Gewicht auf den Aufbau der Applikation gelegt, was dazu führte, dass immer wieder kleinere Probleme beim Entwickeln auftraten. Zum Bsp. gab es Probleme beim automatischen Aktualisieren und übermitteln von Variablen zwischen unterschiedlichen Klassen. Hierfür wurden Handler- und Event-Klassen erstellt, welche die benötigte Funktionalität sicherstellen. Weiter sind beim Erstellen der JUnit Tests Probleme aufgetreten, da Funktionalität und Darstellung nicht getrennt sind und somit die Funktionalität der einzelnen Komponenten nicht mit JUnit getestet werden kann. Dadurch dass Funktionalität und Darstellung nicht getrennt sind, sind die einzelnen Klassen immer noch zu gross und unübersichtlich.

Speichern und Laden eines Stores

Da es mit HTML 5 neuerdings möglich ist, mit Hilfe eine Local Storage grössere Daten (grösser als 4kb bei Cookies) in einem Browser zu speichern, wurde diese Technologie in der GWT Applikation integriert. Beim Speichern eines Stores wird die zu speichernde Datenklasse mit Hilfe von JSON serialisiert und kann als Text im Browser abgelegt werden. Für den Namen des Datenspeichers wird der Name des zu speichernden Stores verwendet.

Will man einen gespeicherten Store abfragen, kann dieser aus einer Auswahl-Liste aller gespeicherten Stores ausgewählt und geladen werden.

Für die Serialisierung und Deserialisierung der zu speicherenden Objekte wird die in der Fassade implementierten Methoden verwendet, da diese die benötigte Funktionalität bereits vorhanden war. Weil die verwendete JSON Library bei GWT leider nicht clientseitig implementiert wurde, muss die Serialisierung und Deserialisierung dadurch serverseitig durchgeführt werden.

Massnahmen

Anpassung Layout

Es wird eine klare Trennung von Eingabe- und Ausgabebereich vorgenommen um eine bessere Übersicht zu schaffen und nicht für Verwirrung zu Sorgen. Dafür wird die Seite neu wie folgt in 3 Bereiche aufgeteilt:

Links, 1. Bereich - Eingabe-Teil: Dieser Bereich ist der Eingabebereich für den Benutzer mit allen Eingabefeldern und Knöpfen, welche für die Erstellung oder für das Abfragen eines Stores benötigt werden. Er soll keine Ausgaben enthalten ausser den Fehlermeldungen, welche den Benutzer auf unzulässige Eingaben hinweisen.

Mitte, 2. Bereich - Grafische Ausgabe: Dieser Bereich ist der grafische Ausgabebereich. Er zeigt dem Benutzer grafisch den Ablauf beim Erstellen und Abfragen eines Stores auf, um die Funktionsweise genauer zu erläutern.

Rechts, 3. Bereich - Anzeige der Ausgaben für die Erklärung der Funktionsweise: Anhand der gewählten Ansicht (5 Sichten), werden in diesem Bereich mehr oder weniger Informationen während des Erstellen oder Abfragen eines Stores angezeigt. Die möglichen Anzeigewerte sind: Polynom, alle Eingaben, Offset, aktueller Schritt (und durchgeführte Schritte), PrimeField...

Beim Vorgang Store Abfragen werden die Knöpfe „Schlüssel kombinieren“, „Kappa Funktion“, „y aus x berechnen“ und „Geheimnis aus y berechnen“ entfernt und durch einen einzigen Knopf „Geheimnis berechnen“ ersetzt. Somit werden auch die Bereiche „Schlüsseleingabe“ und „Geheimnisausgabe“ zu einem Bereich „Geheimnis abfragen“ kombiniert. Für die Eingabe des Werts „Secret Space“ soll der Benutzer nur noch angeben müssen, wie viele Zeichen seine Geheimnisse maximal haben können. Der eigentliche Wert wird Anhand der gewünschten Geheimnislänge berechnet. Die Eingabe des Werts „Security Order“ soll durch einen Slider ersetzt werden. Dieser bietet dem Benutzer einen vordefinierten Bereich, wodurch er eine geführte Wahl des Werts vornehmen kann. Weiter wird hinter jedem Eingabefeld ein Info-Icon mit Mouseover Funktion angezeigt, bei welchem detailliertere Informationen zum Eingabefeld eingeblendet werden können.

MVP Pattern
Um eine saubere Trennung von Funktionalität und Darstellung zu erreichen, wird das Model-View-Presenter Pattern implementiert. Dies macht nicht nur die einzelnen Klassen kleiner, sondern verbessert auch die Testbarkeit der einzelnen Komponenten durch JUnit Tests, welche somit nur noch das Model und den Presenter zu testen benötigen und nicht die View.

D.1.2. Mobile Applikation

Die Applikation wurde auf verschiedenen Android Geräten getestet. Einmal auf dem Emulator des SDK, welcher ein pures Android zur Verfügung stellt. Und weiter wurde die Applikation auch auf einem HTC Desire und einem Samsung Galaxy S getestet. Dabei ist aufgefallen, dass sich die Applikation auf allen Plattformen ein bisschen anders verhält. Vor allem bei der Nutzung des Soft-Keyboard wurden Unterschiede festgestellt. Möchte man die Eingabe über das Soft-Keyboard vereinheitlichen, müsste man eine eigene Keyboard Implementationen zur Verfügung stellen.

Einfache Unit Tests sind rasch implementiert. Sobald es aber darum geht eine Abhängigkeit mit einer zweiten Komponente zu testen, wird es relativ schnell komplex. Möchte man zum Beispiel überprüfen, ob eine andere Activity aufgerufen wird, sobald ein Knopf vom Benutzer gedrückt wird. So ist dies nur schwierig umzusetzen.

Services bieten keinen momentanen Wert für die Implementation. Ob und wie Services, wie in der Analyse vorgesehen, umgesetzt werden, muss im Detail noch genauer untersucht werden. Grundsätzlich ist die Nutzung der KryptonIT Funktionalität von Dritten einfacher, wenn diese die Bibliothek in ihre eigene Applikation einbinden. Weiter wurde herausgefunden, dass die Idee von einem Service im Sinne von Android nicht erfüllt werden kann. Es geht dabei nämlich um Funktionalität, welche im Hintergrund laufen soll, auch wenn sich die dazugehörige Applikation nicht (mehr) im Vordergrund befindet. Wie z.B. ein Musik Player, welcher die Musik auch abspielt, wenn der Benutzer den Web-Browser im Vordergrund hat. Deswegen wurden in der ersten Iteration keine Services umgesetzt. Ansonsten konnte die Funktionalität wie geplant umgesetzt werden.

Für weitere Versionen muss das Benutzer-Erlebnis optimiert werden. Vor allem für die Abfrage von Stores könnten die nötigen Schritte minimiert werden. Es wird angenommen dass das Abfragen eines einzigen Geheimnisses am Meisten verwendet wird. Die Applikation der ersten Iteration ist für diese Anwendung relativ umständlich, da dem Benutzer angeboten wird, mehrere Schlüssel einzugeben. Mit einer Anpassung des Screenflows könnten mehrere Benutzereingaben ausgelassen werden.

Das Persistieren von Stores könnte sowohl in der SQLite Datenbank sowie auch im JSON Format könnte noch effizienter gestaltet werden. Im Moment werden die BigInteger Objekte jeweils in eine Zeichenfolge gespeichert. Hier könnte man sich überlegen, mindestens für die SQLite Datenbank, ob man die einzelnen BigInteger Werte nicht als Binärdaten ablegt.

D.2. Zweite Iteration

D.2.1. Web Applikation

Es wurde festgestellt, dass die zweite Iteration mit zu vielen Tasks überladen wurde und sehr aufwändig war. Dazu waren die einzelnen Tasks nicht immer so schnell erledigt wie geplant. Zum Beispiel, war keine Seekbar (Slider) in GWT vorhanden, was sehr lange dauerte, bis eine lauffähige Seekbar gefunden wurde. Zusätzlich wurde auch in der zweiten Iteration wegen Problemen mit den JUnit Tests wieder wertvolle Zeit verloren. Dies führte dazu, dass noch nicht alles Dokumentiert werden konnte, was noch nachgeholt werden muss. Weitere Erkenntnisse kamen aus der Besprechung mit den Betreuern und dem Experten. Es stellte sich heraus, dass ein normaler Benutzer, welcher die Materie nicht kennt, nicht begreift um was es geht. Deshalb sollen bei den einzelnen Erstellungsschritten detaillierte Informationen eingeblendet werden können. Im Koordinatensystem sollen die Punkte unterschiedlich gefärbt werden (pro Schlüssel-Geheimnis-Paar eine Farbe), um eine Bessere Trennung der Punkte und somit klarere Übersicht zu schaffen. Weiter sollen die Achsenbeschriftungen reduziert werden. Die einzelnen Erstellungsschritte sollen verzögert eingeblendet werden, was das Verständnis für Ablauf verstärkt. Eine weitere Herausforderung wird sein, das Polynom, welches sehr lange werden kann, in einer ansprechenden und übersichtlichen Form darzustellen.

D.2.2. Mobile Applikation

In der zweiten Iteration konnten die wichtigsten Ziele, wie das Redesign der Benutzeroberfläche, hinzufügen weiterer Eingabe-Typen für Schlüssel, Unterstützung von Text-Geheimnissen und viele kleinere Details, umgesetzt werden. Im Bereich Import und Export wurde bereits (im Zusammenhang mit der Nutzung der QR Code Schlüssel Eingabe) eine rudimentäre Import-Export Funktionalität umgesetzt, um Stores zwischen zwei Android Smartphones auszutauschen. In der nächsten Iteration müssen wir uns nochmals auf die Muss Kriterien konzentrieren. In diesem Bereich fehlt noch die Funktion „Kontexbezogene Eingabe simulieren“ und Teile der Funktion „Import und Export von Secret-Stores“. An möglichen Kann-Kriterien und weiteren, während der Arbeit entstandenen Ideen fehlt es nicht. Es ist aber wichtig für die nächste und letzte Iteration eine Priorisierung der Arbeitspakete vorzunehmen, so dass die dritte Iteration bis Ende dieses Jahres abgeschlossen werden.

D.3. Dritte Iteration

D.3.1. Web Applikation

In der dritten Iteration wurden eher allgemeine Erkenntnisse gemacht. Es wurde festgestellt, das es Sinnvoll gewesen wäre, sich zwischendurch vom Programmieren zu lösen, um das Entwickelte von oben her zu betrachten und sich zu überlegen, was optimiert werden könnte, um einen klareren und einfacheren Aufbau zu erstellen. Leider wurde dies aus zeitlichen Gründen während des Projekts nur einmal gemacht. Auch das regelmässige Schreiben der JavaDoc wäre angebracht gewesen, weil das Projekt so stark gewachsen ist, dass die vielen Klassen am Schluss kaum mehr bewältigbar waren.

Würde mehr Zeit oder eine weitere Iteration zur Verfügung stehen, würden wir folgende Dinge noch vorsehen.

Ein Refactoring der Applikation, wobei vorallem folgende Punkte erledigt werden müssten:

- Die Handler Klassen, welche momentan alles Inner-Klassen erstellt wurden, müssten in eigene Klassen ausgebaut werden.
- Die Event Klassen auf nur eine Event Klasse mit mehreren Methoden reduzieren.
- Die Interfaces der Widgets und Views aus den Presenter Klassen in eigene Klassen extrahieren.
- Die Presenter der Widgets auf Presenter pro View reduzieren.
- Aus der Hauptansicht, die Seiten Store erstellen und Store abfragen in eigene Views mit eigenen Presentern extrahieren.
- Die DialogBox Klassen für die Hilfe Texte auf 1 DialogBox mit dynamischem Inhalt reduzieren.

D.3.2. Mobile Applikation

In der dritten Iteration sind die wichtigsten Ziele erreicht worden. Es wurden die Import und Export Funktionen, Asynchrone Erstellung des `SecretSpaceData` in `StoreParametersActivity`, Schlüssel Kombinationen, Store Informations-Maske, das Hilfe-System, die Bearbeitung von eingegebenen Teilschlüssel und viele kleine Details umgesetzt. Einzig die NFC-Eingabe musste aufgrund zu hoher Komplexität und fehlender Test-Hardware (Tags) fallen gelassen werden.

Würde mehr Zeit oder eine weitere Iteration zur Verfügung stehen, würden wir folgende Dinge noch vorsehen. Die NFC-Eingabe könnte umgesetzt werden, wenn NDEF formatierbare Tags zur Verfügung stehen. Im Bereich des Import und Export von Stores mit QR-Codes wäre es interessant die die Base64 Kodierung zu entfernen. So könnten der Overhead der Übertragung reduziert werden und damit auch grössere Stores (höhere Sicherheitsparameter und mehr zufällige Punkte) übertragen werden.

Eine grundsätzliche Änderung würden wir bei der Reihenfolge der Eingabe von Schlüssel und Geheimnis durchführen (bei der Store Erstellung). Diese würden wir so anpassen, dass zuerst die Geheimnisse und dann die Schlüssel eingegeben werden (obwohl die Berechnungsschritte der kryptographische Methode genau andersrum funktionieren). So wäre dann ein neuer Eingabe-Typ „Geheimnis-Eingabe“ umsetzbar, wo der Benutzer ein bereits Eingegebenes Geheimnis als Schlüssel auswählen kann. Durch die Änderung ergeben sich aber auch Sicherheits-relevante Probleme. Da die Geheimnisse länger im Speicher der Applikation aufbewahrt werden müssen.

E. Testfälle

In diesem Kapitel sind alle Testfälle definiert. Dabei wird zwischen automatisierten und Benutzertests unterschieden.

E.1. Fassade

Da die Fassade keine Benutzeroberfläche besitzt, wurden ausschliesslich Unit Tests implementiert.

- `MockObjects.java` : Enthält keine Tests, sondern dient zum Erstellen von arbeitsfähigen KryptonIT Fassade Objekten. Diese werden in verschiedensten Tests wo Abhängigkeiten mit den Fassade Objekten vorhanden sind.
- `TestDeflaterSize.java`
  - `testDeflaterSize` : Weniger ein Test, mehr ein Vergleich der Serializierung von JSON gegenüber geziptem JSON.
- `TestJson.java`
  - `testSecretJson` : Testet die JSON Serializierung und Deserializierung für den Typ `SecretData`.
  - `testKappaJson` : Testet die JSON Serializierung und Deserializierung für den Typ `KappaData`.
  - `testSecretStoreJson` : Testet die JSON Serializierung und Deserializierung für den Typ `SecretStoreData`.
  - `testPolynomJson` : Testet die JSON Serializierung und Deserializierung für den Typ `PolynomData`.
- `TestZipedJson.java`
  - `testSecretJson` : Testet die gezipte JSON Serializierung und Deserializierung für den Typ `SecretData`.
  - `testKappaJson` : Testet die gezipte JSON Serializierung und Deserializierung für den Typ `KappaData`.
  - `testSecretStoreJson` : Testet die gezipte JSON Serializierung und Deserializierung für den Typ `SecretStoreData`.
  - `testPolynomJson` : Testet die gezipte JSON Serializierung und Deserializierung für den Typ `PolynomData`.
- `TestSecretConversion.java`

  - `testSecretBaseConversion` : Testet den Zahlenbasiswechsel, welcher für die Konvertierung von Zahl zu Text verwendet wird.
  - `testSecretConversion` : Testet die Konvertierung von Zahlen- zu Text-Geheimnissen.
  - `testSecretValidationRegex` : Testet den regulären Ausdruck für die Validierung von Text-Geheimnissen.

- `TestStoreManager.java`
  - `testNoRandomPointsStore` : Testet das Erstellen eines Stores mit 0 zufälligen Stützpunkten.
  - `testSigmaBackAndForth` : Testet die KryptonIT Sigma-Funktion.
  - `testCreateAndRetrieveNumberStore` : Testet das Erstellen und Abfragen eines Store mit dem Geheimnis-Typ „Nummer“.
  - `testCreateAndRetrieveTextStore` : Testet das Erstellen und Abfragen eines Store mit dem Geheimnis-Typ „Text“.
  - `testCombiningKeys` : Testet die Kombination (Und-Verknüpfung) von Teilschlüsseln.

E.2. Web Applikation

E.2.1. Benutzer Tests

Applikation aufrufen

Anweisung:	Rufe die Applikation im Browser auf.
Erwartetes Resultat:	Die Applikation wird geladen. Man befindet sich im Tab „Willkommen“.

Tabwechsel nach Store erstellen

Anweisung:	Klicke auf den Tab „Store erstellen“.
Erwartetes Resultat:	Man befindet sich nun im Tab „Store erstellen“ und der Bereich Finiter Körper ist aufgeklappt.

Ansicht wechseln

Anweisung:	Klicke neben dem Titel „Store erstellen“ auf den Radio Button „Erweiterte Ansicht“.
Erwartetes Resultat:	Der Zeichnungsbereich mit 4 Achsen wird eingeblendet.

Finiter Körper erstellen

Anweisung: Wähle die gewünschte Art der Geheimnisse. Definiere die Geheimnislänge über den Slider. Überprüfe, ob, die Sicherheit bereits automatisch auf „normal“ gesetzt wurde. Klicke danach auf den Knopf „Finiter Körper berechnen“.

Erwartetes Resultat: Die Achsen s, x, y wurden beschriftet. Klickt man mit der Maus auf eine Linie der Achsen, so wird ein Klick-Effekt ausgelöst, welcher den dezimalen Wert der Beschriftung anzeigt. Rechts neben dem Koordinatensystem werden die Informationen zum Finiten Körper eingeblendet.

Schlüsseleingabe erster Schlüssel

Anweisung: Wechsle auf den Bereich Schlüsseleingabe, um diesen zu öffnen. Klicke auf den Link „Teilschlüssel hinzufügen“. Gib etwas bei den zwei Teilschlüsseln ein.

Erwartetes Resultat: Ein zweites Eingabefeld zum ersten Schlüssel wurde eingeblendet und konnte abgefüllt werden.

Schlüsseleingabe zweiter Schlüssel

Anweisung: Klicke auf den Link „Schlüssel hinzufügen“. Gib einen 2. Schlüssel ein.

Erwartetes Resultat: Ein drittes Eingabefeld für die Eingabe des zweiten Schlüssels wurde eingeblendet und konnte abgefüllt werden.

Schlüsseleingabe abschliessen

Anweisung: Klicke auf den Knopf „Kappa Funktion“.

Erwartetes Resultat: Die k-Achse wird beschriftet. Die eingegebenen Schlüssel werden auf der k-Achse angezeigt. Es wird eine gestrichelte Linie mit einem K (Kappa) eingeblendet und nach einer kurzen Verzögerung werden die transformierten Schlüssel auf der x-Achse angezeigt. Klickt man mit der Maus auf eine Linie der Achsen, so wird ein Klick-Effekt ausgelöst, welcher den dezimalen Wert der Beschriftung anzeigt. Klickt man mit der Maus auf einen der Punkte, wird ein Klick-Effekt ausgelöst, welcher den Wert des Punktes anzeigt. Rechts neben dem Koordinatensystem werden die Informationen zum erstellten Kappa-Data Objekt mit den Schlüssel-Werten und den transformierten x-Werten der Punkte und dem Offset eingeblendet.

Geheimniseingabe

Anweisung: Wechsle auf den Bereich Geheimniseingabe, um diesen zu öffnen.

Erwartetes Resultat: Es müssen zwei Eingabefelder vorhanden sein. (Pro Schlüssel ein Geheimnis)

Geheimniseingabe 2

Anweisung: Gib in beide Eingabefelder einen Wert ein, welcher nicht länger ist, als bei der Eingabe „Maximale Anzahl Stellen“ definiert wurde. Klicke auf den Knopf „Sigma Funktion“.

Erwartetes Resultat: Die eingegebenen Schlüssel werden auf der s-Achse angezeigt. Es wird eine gestrichelte Linie mit einem E (Sigma) eingeblendet und nach einer kurzen Verzögerung werden die transformierten Schlüssel auf der y-Achse angezeigt. Klickt man mit der Maus auf einen der Punkte, wird ein Klick-Effekt ausgelöst, welcher den Wert des Punktes anzeigt. Rechts neben dem Koordinatensystem werden die Werte und die transformierten Werte der Punkte der Geheimnisse eingeblendet. Nach einer weiteren kurzen Verzögerung werden die Punkte, welche sich aus den Punkten von der x- und y-Achse ergeben, im Koordinatensystem angezeigt. Klickt man mit der Maus auf einen der Punkte, wird ein Klick-Effekt ausgelöst, welcher den x- und y-Wert des Punktes anzeigt.

Polynominterpolation

Anweisung: Wechsle auf den Bereich Polynominterpolation, um diesen zu öffnen. Gib eine Zahl beim Eingabefeld Anzahl Zufallspunkte ein. Klicke auf den Knopf „Zufallspunkte generieren“.

Erwartetes Resultat: Die definierte Anzahl Zufallspunkte werden generiert und im Koordinatensystem angezeigt. Es müssen so viele Zufallspunkte vorhanden sein, wie definiert wurde. Klickt man mit der Maus auf einen der Punkte, wird ein Klick-Effekt ausgelöst, welcher den x- und y-Wert des Punktes anzeigt. Rechts neben dem Koordinatensystem werden die Werte der generierten Zufallspunkte eingeblendet.

Polynominterpolation 2

Anweisung: Klicke auf den Knopf „Polynom berechnen“.

Erwartetes Resultat: Man erhält eine Meldung, dass der Store erstellt wurde. Rechts neben dem Koordinatensystem wird das erstellte Polynom und das erstellte Chiffrat (Secret Store Data) eingeblendet.

Store speichern

Anweisung:	Wechsle auf den Bereich Store speichern, um diesen zu öffnen. Gib einen Namen für den Store im Eingabefeld „Store Name“ ein. Klick auf den Knopf „Store speichern“.
Erwartetes Resultat:	Es wird eine Meldung angezeigt, dass der Store gespeichert wurde.

Store speichern 2

Anweisung:	Klicke auf den Knopf „Clear Graphic“ Data.
Erwartetes Resultat:	Das Koordinaten System und die Achsen werden wieder auf den Ursprungszustand zurückgesetzt. (Keine Punkte und Achsenbeschriftungen mehr.) Der Informationsbereich auf der rechten Seite wird ausgeblendet.

Tabwechsel nach Store laden

Anweisung:	Klicke auf den Tab „Store abfragen“.
Erwartetes Resultat:	Man befindet sich im Tab „Store abfragen“ und der Bereich Store laden ist aufgeklappt.

Ansicht wechseln

Anweisung:	Klicke neben dem Titel „Store abfragen“ auf den Radio Button „Angreifer Ansicht 4“.
Erwartetes Resultat:	Der Zeichnungsbereich mit 4 Achsen wird eingeblendet.

Store laden

Anweisung:	Öffne das DropDown Feld und wähle den zuvor gespeicherten Store aus. Klicke auf den Knopf „Store laden“.
Erwartetes Resultat:	Die Achsen s, x, y werden beschriftet. Klickt man mit der Maus auf eine Linie der Achsen, so wird ein Klick-Effekt ausgelöst, welcher den dezimalen Wert der Beschriftung anzeigt. Im Ausgabefeld unter dem Knopf wird der Name des geladenen Stores ausgegeben. Rechts neben dem Koordinatensystem werden die Informationen zum geladenen Store eingeblendet.

Geheimnis abfragen

Anweisung: Wechsle auf den Bereich Store abfragen, um diesen zu öffnen. Gib einen der Schlüssel ein, welcher beim Store erstellen definiert wurde. (Füge falls nötig, zusätzliche Teilschlüssel hinzu.) Klicke auf den Knopf „Kappa Funktion“.

Erwartetes Resultat: Die k-Achse wird beschriftet. Der eingegebene Schlüssel wird auf der k-Achse angezeigt. Es wird eine gestrichelte Linie mit einem K (Kappa) eingeblendet und nach einer kurzen Verzögerung wird der transformierte Schlüssel auf der x-Achse angezeigt. Fährt man mit der Maus auf eine Linie der Achsen, so wird ein Klick-Effekt ausgelöst, welcher den dezimalen Wert der Beschriftung anzeigt. Klickt man mit der Maus auf einen der Punkte, wird ein Klick-Effekt ausgelöst, welcher den Wert des Punktes anzeigt. Rechts neben dem Koordinatensystem wird das erstellte Kappa-Data Objekt mit dem x-Wert des Punktes und dem Offset eingeblendet. Nach einer weiteren kurzen Verzögerung wird der zugehörige Punkt auf der y-Achse angezeigt. Es wird eine gestrichelte Linie mit einem E (Sigma) eingeblendet. Klickt man mit der Maus auf den Punkt, wird ein Klick-Effekt ausgelöst, welcher den Wert des Punktes anzeigt. Rechts neben dem Koordinatensystem wird der Wert des Punktes ausgegeben. Nach einer erneuten Verzögerung wird der zugehörige Punkt auf der s-Achse angezeigt. Klickt man mit der Maus auf den Punkt, wird ein Klick-Effekt ausgelöst, welcher den Wert des Punktes anzeigt. Im Ausgabefeld unter dem Knopf wird der effektive Wert des Punktes ausgegeben. Dieser Wert sollte dem Geheimnis entsprechen, welches bei der Store-Erstellung eingegeben wurde.

Geheimnis abfragen 2

Anweisung: Klicke auf den Knopf „Clear Graphic Data“.

Erwartetes Resultat: Das Koordinaten System und die Achsen werden wieder auf den Ursprungszustand zurückgesetzt. (Keine Punkte und Achsenbeschriftungen mehr.) Der Informationsbereich auf der rechten Seite wird ausgeblendet.

Tabwechsel nach Import/Export

Anweisung: Klicke auf den Tab „Import/Export“.

Erwartetes Resultat: Man befindet sich im Tab „Import/Export“.

Store exportieren

Anweisung: Wähle den zuvor gespeicherten Store aus dem DropDown aus. Wähle als Export-Typ „QR Code“ aus. Klicke auf den Knopf „Store exportieren“.

Erwartetes Resultat: Der QR Code des Stores wird angezeigt.

Store importieren

Anweisung:	Importiere den am Display angezeigten QR Code mit der Android Applikation.
Erwartetes Resultat:	Store wurde erfolgreich aufs Handy importiert.

Store exportieren 2

Anweisung:	Wähle den zuvor gespeicherten Store erneut aus dem DropDown aus. Wähle als Export-Typ „JSon String“ aus. Klicke auf den Knopf „Store exportieren“.
Erwartetes Resultat:	Es wird ein Ausgabefeld eingeblendet, worin der Store als JSon String angezeigt wird.

Store importieren 2

Anweisung:	Kopiere die Ausgabe (JSon String) des letzten Tests und füge ihn im Eingabefeld bei Store Importieren ein. Ändere den Namen des Stores im JSon String. Klicke auf den Knopf „Store importieren“.
Erwartetes Resultat:	Es wird eine Meldung angezeigt, dass der Store erfolgreich importiert wurde.

Store löschen

Anweisung:	Wähle den importierten Store im DropDown aus. Klicke auf den Knopf „Store löschen“.
Erwartetes Resultat:	Es wird eine Meldung angezeigt, dass der Store gelöscht wurde. Der Store ist in keinem DropDown mehr vorhanden.

E.2.2. Automatisierte Tests

Für die letzte Iteration wurden GWT Unit Tests umgesetzt. Es wurden vier Test Klassen erstellt.

- `WidgetTests.java` : Testet, ob alle Werte der erstellten Widgets gesetzt und abgefragt werden können und überprüft ob die Standard-Werte, wo benötigt, bei den Widgets gesetzt sind.
  - `testFiniteBodyWidgetDefaultValuesSet` : Testet, ob die Standard-Werte im FiniteBodyWidget gesetzt sind.
  - `testKeysWidget` : Testet die Eingabe eines Schlüssels im KeysWidget.
  - `testKeysWidgetWithThreeKeys` : Testet die Eingabe von drei Schlüsseln im KeysWidget.

  - `testKeysWidgetWithTwoKeysWithTwoKeyParts` : Testet die Eingabe von zwei Schlüsseln bestehend aus jeweils zwei Teilschlüsseln im KeysWidget.
  - `testSecretsWidget` : Testet die Eingabe eines Geheimnis im SecretsWidget.
  - `testSecretsWidgetWithThreeSecrets` : Testet die Eingabe von drei Geheimnissen im SecretsWidget.
  - `testPolynomInterpolationWidget` : Testet die Eingabe der Anzahl der zu generierenden Zufallspunkte im PolynomInterpolationWidget.
  - `testSaveStoreWidget` : Testet die Eingabe des Namen des zu speichernden Stores im SaveStoreWidget.
  - `testRetrieveSecretWidget` : Testet die Eingabe eines Schlüssels im RetrieveSecretWidget.
  - `testRetrieveSecretWidgetWithTwoKeyParts` : Testet die Eingabe eines Schlüssels bestehend aus zwei Teilschlüsseln im RetrieveSecretWidget.
  - `testImportWidget` : Testet die Eingabe eines Stores als JSON-String.
  - `testExportWidgetDefaultValueSet` : Testet, ob die Standard-Werte im ExportWidget gesetzt sind.

- `ValidatorTests.java` : Testet die Validierungen. Es gibt Eingabe-Validierungen und Validierungen, welche sicherstellen, dass alle Schritte vor dem aktuellen Schritt durchgeführt wurden.
  - `testFiniteBodyValidatorSecretTypeEmpty` : Testet die Validierung beim Erstellen des Finiten Körpers, wenn kein SecretType gewählt wurde.
  - `testKeysValidatorFiniteBodyEmpty` : Testet die Validierung beim Erstellen der Schlüssel, wenn kein Finiter Körper erstellt wurde.
  - `testKeysValidatorKeyPartEmpty` : Testet die Validierung beim Erstellen der Schlüssel, wenn leere Teilschlüssel eingegeben wurden.
  - `testKeysValidatorKeyPartsIdentic` : Testet die Validierung beim Erstellen der Schlüssel, wenn identische Teilschlüssel eines Schlüssels eingegeben wurden.
  - `testSecretsValidatorFiniteBodyEmpty` : Testet die Validierung beim Erstellen der Geheimnisse, wenn kein Finiter Körper erstellt wurde.
  - `testSecretsValidatorKeysEmpty` : Testet die Validierung beim Erstellen der Geheimnisse, wenn keine Schlüssel erstellt wurden.
  - `testSecretsValidatorSecretEmpty` : Testet die Validierung beim Erstellen der Geheimnisse, wenn leere Geheimnisse eingegeben wurden.
  - `testSecretsValidatorOnlyNumbers` : Testet die Validierung beim Erstellen der Geheimnisse, wenn keine Zahlen eingeben wurden und nur Zahlen als Geheimnis eingegeben werden darf.

- `testSecretsValidatorOnlySigns` : Testet die Validierung beim Erstellen der Geheimnisse, wenn nicht erlaubte Zeichen eingegeben wurden und Text als Geheimnis eingegeben werden darf.
- `testSecretsValidatorTooLong` : Testet die Validierung beim Erstellen der Geheimnisse, wenn ein Geheimnis zu lange für den definierten finiten Körper ist.
- `testPolynomInterpolationValidatorFiniteBodyEmpty` : Testet die Validierung beim Erstellen der Zufallspunkte, wenn kein Finiter Körper erstellt wurde.
- `testPolynomInterpolationValidatorFiniteBodyEmpty2` : Testet die Validierung beim Berechnen der PolynomInterpolation, wenn kein Finiter Körper erstellt wurde.
- `testPolynomInterpolationValidatorKeysEmpty` : Testet die Validierung beim Erstellen der Zufallspunkte, wenn keine Schlüssel erstellt wurden.
- `testPolynomInterpolationValidatorKeysEmpty2` : Testet die Validierung beim Berechnen der PolynomInterpolation, wenn keine Schlüssel erstellt wurden.
- `testPolynomInterpolationValidatorSecretsEmpty` : Testet die Validierung beim Erstellen der Zufallspunkte, wenn keine Geheimnisse erstellt wurden.
- `testPolynomInterpolationValidatorSecretsEmpty2` : Testet die Validierung beim Berechnen der PolynomInterpolation, wenn keine Geheimnisse erstellt wurden.
- `testPolynomInterpolationValidatorRandomPointsEmpty` : Testet die Validierung beim Berechnen der PolynomInterpolation, wenn keine Zufallspunkte erstellt wurden.
- `testPolynomInterpolationValidatorRandomPointFieldEmpty` : Testet die Validierung beim Erstellen der Zufallspunkte, wenn das Eingabefeld für die Anzahl Zufallspunkte leer.
- `testPolynomInterpolationValidatorRandomPointFieldNotNumber` : Testet die Validierung beim Erstellen der Zufallspunkte, wenn keine Zahl in das Eingabefeld für die Anzahl Zufallspunkte eingegeben wurde.
- `testSaveStoreValidatorStoreEmpty` : Testet die Validierung beim Speichern eines Stores, wenn noch kein Store erstellt wurde.
- `testSaveStoreValidatorStoreNameEmpty` : Testet die Validierung beim Speichern eines Stores, wenn kein Name für den zu speichernden Store eingegeben wurde.
- `testSaveStoreValidatorStoreNameInvalid` : Testet die Validierung beim Speichern eines Stores, wenn der Name für den zu speichernden Store unerlaubte Zeichen beinhaltet.
- `testLoadStoreValidatorNoStoreSelected` : Testet die Validierung beim Laden eines Stores, wenn kein Store zum Laden ausgewählt wurde und auch kein temporärer Store geladen ist.
- `testRetrieveSecretValidatorNoStoreLoaded` : Testet die Validierung beim Abfragen eines Geheimnisses, wenn kein Store zum Abfragen geladen wurde.
- `testRetrieveSecretValidatorKeyPartEmpty` : Testet die Validierung beim Abfragen eines Geheimnisses, wenn leere Teilschlüssel eingegeben wurden.

  - `testRetrieveSecretValidatorKeyPartsIdentic` : Testet die Validierung beim Abfragen eines Geheimnisses, wenn identische Teilschlüssel eingegeben wurden.

  - `testImportExportValidatorImportEmpty` : Testet die Validierung beim Importieren eines Stores, wenn kein Store (JSON String) eingegeben wurde.

  - `testImportExportValidatorImportInvalid` : Testet die Validierung beim Importieren eines Stores, wenn ein ungültiges Store Format (JSON String) eingegeben wurde.

  - `testImportExportValidatorExportNoStoreSelected` : Testet die Validierung beim Exportieren eines Stores, wenn kein Store zum Exportieren ausgewählt wurde.

  - `testImportExportValidatorExportNoExportTypeSelected` : Testet die Validierung beim Exportieren eines Stores, wenn der ExportType des zu exportierenden Stores nicht ausgewählt wurde.

  - `testImportExportValidatorDeleteNoStoreSelected` : Testet die Validierung beim Löschen eines Stores, wenn kein Store zum Löschen ausgewählt wurde.

- `KryptonITAsyncTests.java` : Testet die Methoden des KryptonIT Service. Die Methoden welche auf dem Server ausgeführt werden, werden vom Client aufgerufen (RPC-Methoden). Jeder Test muss warten, bis die Antwort vom Server an den Client zurückgegeben wurde.

  - `testGetSpaceCreateDataAsync` : Tested die getSpace() Methode, welche den finiten Körper (SpaceCreateData) beim Erstellen eines Stores berechnet und dem Client zurück gibt.

  - `testGetKappaDataAsync` : Testet die getKappa() Methode, welche das KappaData Objekt (mit den Schlüsseln) beim Erstellen eines Stores berechnet und dem Client zurück gibt.

  - `testGetSigmaToYAsync` : Testet die getSigmaToY() Methode, welche die y-Werte aus den Geheimnis-Werten beim Erstellen eines Stores berechnet und dem Client zurück gibt.

  - `testGetKappa2Async` : Testet die getKappa2() Methode, welche das KappaData Objekt (mit dem Schlüssel) beim Abfragen eines Stores berechnet und dem Client zurück gibt.

  - `testGetYAsync` : Testet die getY() Methode, welche den zugehörigen y-Wert eines x-Werts beim Abfragen eines Stores berechnet und dem Client zurück gibt.

  - `testGetSigmaToSecretAsync` : Testet die getSigmaToSecret() Methode, welche den Geheimnis-Wert eines y-Werts beim Abfragen des Stores berechnet und dem Client zurück gibt.

  - `testGetJsonStringAsync` : Testet die getJsonString() Methode, welche den JSON String eines Stores für das Exportieren erstellt und dem Client zurück gibt.

  - `testCompressedJsonStringAsync` : Testet die getCompressedJsonString() Methode, welche den komprimierten String eines Stores für das Exportieren als QR-Code erstellt und dem Client zurück gibt.

  - `testGetCombinedKeyAsync` : Testet die getCombinedKey() Methode, welche die Teilschlüssel beim Erstellen eines Stores kombiniert und dem Client als einen Schlüssel zurück gibt.

  - `testIsValidTextSecretAsync` : Testet die isValidTextSecret() Methode, welche überprüft, ob die eingegeben Geheimnisse beim Erstellen eines Stores nicht erlaubte Zeichen beinhalten und dem Client das Resultat (true oder false) zurück gibt.

  - `testGetConvertedSecretListAsync` : Testet die getConvertedSecretList() Methode, welche die Geheimnis-Werte von den eingegeben Geheimnissen bei der Erstellung eines Stores berechnet und dem Client zurück gibt.

  - `testGetDeconvertedSecretAsync` : Testet die getDeconvertedSecret() Methode, welche ein Geheimnis vom Geheimnis-Wert beim Abfragen eines Stores berechnet und dem Client zurück gibt.

  - `testGetStoreFromJSonStringAsync` : Testet die getStore() Methode, welche einen Store (SecretStoreData Objekt) aus einem JSON-String beim Importieren erstellt und dem Client zurück gibt.

  - `testGetStoreAsync` : Testet die getStore() Methode, welche einen Store (SecretStoreData Objekt) beim Erstellen eines Stores berechnet und dem Client zurück gibt.

- `KryptonITRPCTests.java` : Testet die einzelnen Schritte beim Erstellen und Abfragen eines Stores.

  - `testKeysWidget_withRPC` : Testet, ob Keys erstellt werden.

  - `testSecretsWidget_withRPC` : Testet, ob Secrets erstellt werden.

  - `testPolynomInterpolationWidget_withRPC` : Testet, ob ein Store erstellt wird.

  - `testRetrieveSecretWidget_withRPC` : Testet, ob ein Geheimnis von einem Store abgefragt werden kann und ob das richtige Geheimnis zurückgegeben wird.

E.3. Mobile Applikation

E.3.1. Benutzer Tests

Applikation starten

Anweisung:	Starte die Applikation über den Launcher.
Erwartetes Resultat:	Der oberste Eintrag in der Liste ist „Neuer Store“. Unterhalb werden, falls vorhanden, existierende Stores angezeigt.

Store Erstellen

Neuer Store erstellen

Anweisung:	Wähle den Eintrag „Neuer Store“ aus.
Erwartetes Resultat:	Die Ansicht wird gewechselt und zeigt die Parameter für den neuen Store an.

Store Parameter eingeben

Anweisung:	Gib den Namen für den neuen Store ein. Wähle dann die Anzahl der Geheimnis-Stellen über den Slider aus. Schliesse die Eingabe der Parameter mit der Schaltfläche „Weiter“ ab.
Erwartetes Resultat:	Die Ansicht für die Schlüssel-Eingabe wird angezeigt.

Schlüssel hinzufügen

Anweisung:	Wähle den Eintrag „Schlüssel hinzufügen“ aus. Anschliessend muss der Eingabe-Typ „Freie Eingabe“ ausgewählt werden. Gib eine beliebige Eingabe in das Textfeld ein und schliesse die Schlüssel Eingabe mit „Ok“ ab.
Erwartetes Resultat:	Der Schlüssel wurde in die Liste hinzugefügt.

Teil-Schlüssel hinzufügen

Anweisung:	Klappe den vorher erstellten Schlüssel auf, indem du diesen anklickst. Dann wähle den Eintrag „Teilschlüssel hinzufügen“ aus. Selektiere den Eingabe-Typ „Freie Eingabe“ aus. Bestätige den Teilschlüssel mit „Ok“.
Erwartetes Resultat:	Der Teilschlüssel wurde dem korrekten Schlüssel hinzugefügt.

Zweiten Schlüssel hinzufügen

Anweisung: Wähle den Eintrag „Schlüssel hinzufügen“ aus. Anschliessend muss der Eingabe-Typ „Freie Eingabe“ ausgewählt werden. Gib eine beliebige Eingabe in das Textfeld ein und schliesse die Schlüssel Eingabe mit „Ok“ ab.

Erwartetes Resultat: Der Schlüssel wurde in die Liste hinzugefügt.

Schlüssel Eingabe abschliessen

Anweisung: Schliesse die Eingabe der Schlüssel mit der Schaltfläche „Weiter“ ab.

Erwartetes Resultat: Die Ansicht für die Eingabe der Geheimnisse wird angezeigt. Es müssen zwei Geheimnisse in der Liste aufgeführt werden.

Geheimnis eingeben

Anweisung: Wähle das erste Geheimnis in der Liste aus und gib einen Wert ein, welchen du mit „Ok“ bestätigst.

Erwartetes Resultat: Das erste Geheimnis wird mit einem grünen Häkchen angezeigt.

Zweites Geheimnis eingeben

Anweisung: Wähle das zweite Geheimnis in der Liste aus. Gib im erscheinenden Dialog einen Wert ein und bestätige diese mit „Ok“.

Erwartetes Resultat: Das zweite Geheimnis wird mit einem grünen Häkchen angezeigt.

Store Erstellung abschliessen

Anweisung: Schliesse die Erstellung mit der Schaltfläche „Abschliessen“ ab.

Erwartetes Resultat: Die Liste der Stores wird angezeigt und enthält den eben erstellten Store.

Store abfragen

Store auswählen

Anweisung: Wähle in der Liste einen vorhanden Store aus.

Erwartetes Resultat: Die Auflistung der Eingabe-Typen wird angezeigt.

Schlüssel hinzufügen

Anweisung:	Wähle den Eintrag „Freie Eingabe“ aus. Gib eine beliebige Eingabe in das Textfeld ein und schliesse die Schlüsseleingabe mit „Ok“ ab.
Erwartetes Resultat:	Der Schlüssel wird in die Liste hinzugefügt.

Schlüssel Eingabe abschliessen

Anweisung:	Schliesse die Eingabe der Schlüssel mit der Schaltfläche „Weiter“ ab.
Erwartetes Resultat:	Die Ansicht für die Anzeige der Geheimnisse wird angezeigt. Es ist genau ein Geheimnis aufgeführt.

Store Abfrage abschliessen

Anweisung:	Schliesse die Abfrage mit der Schaltfläche „Abschliessen“ ab.
Erwartetes Resultat:	Die Liste der Stores wird angezeigt.

E.3.2. Automatisierte Tests

Die automatisierten Test wurden mit JUnit in einem separaten Test-Projekt umgesetzt. Sie sind in Unit Tests und Blackbox Tests aufgeteilt.

Unit Test

Für jede Komponente wurde eine separate Unit Test Klasse erstellt. Die Test beschränken sich auf die Überprüfung ob die Activity fehlerfrei erstellt werden kann und ob alle benötigten UI Elemente vorhanden sind.

- `InputFreeActivityTest.java`
  - `testOrientation` : Testet ob die Activity bei einem Wechsel der Bildschirm-Orientierung den Zustand behält.
  - `testPreconditions` : Testet ob die Activity erstellt werden kann und die UI Elemente vorhanden sind.
- `InputlistActivityTest.java`
  - `testOrientation` : Testet ob die Activity bei einem Wechsel der Bildschirm-Orientierung den Zustand behält.
  - `testPreconditions` : Testet ob die Activity erstellt werden kann und die UI Elemente vorhanden sind.
- `KeyInputActivityTest.java`

  - `testOrientation` : Testet ob die Activity bei einem Wechsel der Bildschirm-Orientierung den Zustand behält.
  - `testPreconditions` : Testet ob die Activity erstellt werden kann und die UI Elemente vorhanden sind.
  - `testKeyList` : Testet ob in der Liste der Keys das erste Element, zum hinzufügen von Keys, vorhanden ist. .

- `SecretInputActivityTest.java`
  - `testOrientation` : Testet ob die Activity bei einem Wechsel der Bildschirm-Orientierung den Zustand behält.
  - `testPreconditions` : Testet ob die Activity erstellt werden kann und die UI Elemente vorhanden sind.

- `SecretRetrieveActivityTest.java`
  - `testOrientation` : Testet ob die Activity bei einem Wechsel der Bildschirm-Orientierung den Zustand behält.
  - `testPreconditions` : Testet ob die Activity erstellt werden kann und die UI Elemente vorhanden sind.

- `StoreListActivityTest.java`
  - `testOrientation` : Testet ob die Activity bei einem Wechsel der Bildschirm-Orientierung den Zustand behält.
  - `testPreconditions` : Testet ob die Activity erstellt werden kann und die UI Elemente vorhanden sind.
  - `testStoreList` : Testet ob in der Liste der Stores das erste Element, zum hinzufügen von Stores, vorhanden ist.

- `StoreListActivityTest.java`
  - `testDefaults` : Testet ob die Standardwerte beim Laden der Activity richtig gesetzt sind.
  - `testDigitsSeekBar` : Testet ob der Schieberegler für die Anzahl der Stellen das UI richtig aktualisiert.
  - `testOrderSeekBar` : Testet ob der Schieberegler für den Sicherheitsparameter 1 das UI richtig aktualisiert.
  - `testFactorSeekBar` : Testet ob der Schieberegler für den Sicherheitsparameter 2 das UI richtig aktualisiert.
  - `testSecurityNormal` : Testet ob die Werte für die Garnitur „Normal“ richtig gesetzt werden.

  - `testSecurityHigh` : Testet ob die Werte für die Garnitur „Hoch“ richtig gesetzt werden.
  - `testSecurityInsane` : Testet ob die Werte für die Garnitur „Verrückt“ richtig gesetzt werden.
  - `testMissingName` : Testet ob der Name zwingend eingegeben muss.
  - `testWrongMinMaxPoints` : Testet ob die Anzahl der zufälligen Punkte validiert werden.
  - `testOrientationChangeMainTab` : Testet ob das Register „Allgemein“ bei einem Wechsel der Bildschirm-Orientierung den Zustand behält.
  - `testOrientationChangeAdvancedTab` : Testet ob das Register „Erweitert“ bei einem Wechsel der Bildschirm-Orientierung den Zustand behält.
  - `testPreconditions` : Testet ob die Activity erstellt werden kann und die UI Elemente vorhanden sind.

- `SecretStoreProviderTest.java`
  - `testPreconditions` : Testet ob der Content Provider instanziiert werden kann.
  - `testDeleteSingleRecord` : Testet ob ein einzelner Store gelöscht werden kann.
  - `testInsertAndSelect` : Testet ob ein Store eingefügt und wieder geladen werden kann.
  - `testQuery` : Testet ob eine Liste von Stores abgefragt werden kann.
  - `testUpdateSingleRecord` : Testet ob ein Store aktualisiert werden kann.
  - `testUpdateMultiple` : Testet ob das Aktualisieren von mehreren Stores zu einem Fehler führt.
  - `testDeleteMultiple` : Testet ob das Löschen von mehreren Stores zu einem Fehler führt.

Blackbox Test

Für die Blackbox Tests wurde das Projekt Robotium verwendet. Die Tests wurden in der Test-Klasse `RobotiumTest.java` umgesetzt.

- `testAddStore` Erstellt einen Store mit dem Namen „roboter <Nummer>“. Welcher die zwei Geheimnisse „I love it“ und „Xsf93bla“ enthält.
- `testQueryStoreSimple` Fragt den letzten Store in der Store-Liste mit den Schlüsseln aus dem Test `testAddStore` im „Einfachen“-Modus ab und überprüft ob das erste Geheimnis korrekt berechnet wurden.
- `testQueryStoreExtended` Fragt den letzten Store in der Store-Liste mit den Schlüsseln aus dem Test `testAddStore` im „Mehfrach“-Modus ab und überprüft ob die Geheimnisse korrekt berechnet wurden.

F. Testresultate

F.1. 1. Iteration

Hier sind die Resultate der im Kapitel E definierten Tests enthalten.

F.1.1. Web Applikation

Durchgeführt von Louis Bernath am 12.11.2011 für Iteration 1.

Testfall	Ergebnis
Applikation aufrufen	Ok
Finiter Körper erstellen	Ok
Schlüsseleingabe	Ok
Schlüsseleingabe 2	Ok
Geheimniseingabe	Ok
Geheimniseingabe 2	Ok
Polynominterpolation	Ok
Polynominterpolation 2	Ok
Polynominterpolation 3	Ok
Store speichern	Ok
Store speichern 2	Ok
Store laden	Ok
Store laden 2	Ok
Schlüsseleingabe	Ok
Geheimnisausgabe	Ok
Geheimnisausgabe 2	Ok
Geheimnisausgabe 3	Ok

F.1.2. Mobile Applikation

Durchgeführt von Jan Liechti am 13.11.2011 für Iteration 1.

Testfall	Ergebnis
Applikation starten	Ok
Neuer Store erstellen	Ok
Store Parameter eingeben	Ok
Schlüssel hinzufügen	Ok
Null Schlüssel hinzufügen	Ok
Schlüssel Eingabe abschliessen	Ok
Geheimnis eingeben	Ok
Zweites Geheimnis eingeben	Ok
Store Erstellung abschliessen	Ok
Schlüssel hinzufügen	Ok
Schlüssel Eingabe abschliessen	Ok
Store Abfrage abschliessen	Ok

JUnit Testresultate

Test results

12.11.2011

Summary:

Total 17 tests in 9 classes.
Total errors: 0
Total failures: 0
Tests succeded: 17
Total time: 46.524 s

Successes

* Class ch.bfh.kryptonIT.android.test.SecretRetrieveActivityTest : Test testPreconditions finished successfully in 1.650 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testDeleteSingleRecord finished successfully in 0.350 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testInsertAndSelect finished successfully in 0.250 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testPreconditions finished successfully in 0.150 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testQuery finished successfully in 0.200 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testUpdateSingleRecord finished successfully in 0.025 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testAndroidTestCaseSetupProperly finished successfully in 0.075 s.
* Class ch.bfh.kryptonIT.android.test.SecretInputActivityTest : Test testPreconditions finished successfully in 1.025 s.
* Class ch.bfh.kryptonIT.android.test.StoreDetailsActivityTest : Test testPreconditions finished successfully in 1.025 s.
* Class ch.bfh.kryptonIT.android.test.RobotiumTest : Test testAddStore finished successfully in 21.676 s.
* Class ch.bfh.kryptonIT.android.test.RobotiumTest : Test testQueryStore finished successfully in 13.851 s.
* Class ch.bfh.kryptonIT.android.test.InputlistActivityTest : Test testPreconditions finished successfully in 1.075 s.
* Class ch.bfh.kryptonIT.android.test.KeyInputActivityTest : Test testKeyList finished successfully in 1.025 s.

* Class ch.bfh.kryptonIT.android.test.KeyInputActivityTest : Test testPreconditions finished successfully in 0.775 s.
* Class ch.bfh.kryptonIT.android.test.InputFreeActivityTest : Test testPreconditions finished successfully in 1.222 s.
* Class ch.bfh.kryptonIT.android.test.StoreListActivityTest : Test testPreconditions finished successfully in 1.050 s.
* Class ch.bfh.kryptonIT.android.test.StoreListActivityTest : Test testStoreList finished successfully in 1.100 s.

F.1.3. Fassade

Test results

12.11.2011

Summary:

Total 6 tests in 2 classes.
Total errors: 0
Total failures: 0
Tests succeded: 6
Total time: 0.645 s

Successes

* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testSecretJson finished successfully in 0.001 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testKappaJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testSecretStoreJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestKryptex : Test testNoRandomPointsStore finished successfully in 0.630 s.
* Class ch.bfh.kryptonIT.facade.test.TestKryptex : Test testSigmaBackAndForth finished successfully in 0.001 s.
* Class ch.bfh.kryptonIT.facade.test.TestKryptex : Test testCreateAndRetrieveStore finished successfully in 0.013 s.

F.2. 2. Iteration

Hier sind die Resultate der im Kapitel E definierten Tests enthalten.

F.2.1. Web Applikation

Durchgeführt von Louis Bernath am 12.12.2011 für Iteration 2.

Testfall	Ergebnis
Applikation aufrufen	Ok
Finiter Körper erstellen	Ok
Schlüsseleingabe erster Schlüssel	Ok
Schlüsseleingabe zweiter Schlüssel	Ok
Schlüsseleingabe abschliessen	Ok
Geheimniseingabe	Ok
Geheimniseingabe 2	Ok
Polynominterpolation	Ok
Polynominterpolation 2	Ok
Store speichern	Ok
Store speichern 2	Ok
Store laden	Ok
Store laden 2	Ok
Geheimnis abfragen	Ok
Geheimnis abfragen 2	Ok

F.2.2. Mobile Applikation

Durchgeführt von Jan Liechti am 12.12.2011 für Iteration 2.

Testfall	Ergebnis
Applikation starten	Ok
Neuer Store erstellen	Ok
Store Parameter eingeben	Ok
Schlüssel hinzufügen	Ok
Teil-Schlüssel hinzufügen	Ok
Zweiten Schlüssel hinzufügen	Ok
Schlüssel Eingabe abschliessen	Ok
Geheimnis eingeben	Ok
Zweites Geheimnis eingeben	Ok
Store Erstellung abschliessen	Ok
Store auswählen	Ok
Schlüssel hinzufügen	Ok
Schlüssel Eingabe abschliessen	Ok
Store Abfrage abschliessen	Ok

JUnit Testresultate

Test results

08.12.2011

Summary:

Total 35 tests in 9 classes.
Total errors: 0
Total failures: 0
Tests succeded: 35
Total time: 84.249 s

Successes

* Class ch.bfh.kryptonIT.android.test.SecretRetrieveActivityTest : Test testOrientation finished successfully in 1.625 s.
* Class ch.bfh.kryptonIT.android.test.SecretRetrieveActivityTest : Test testPreconditions finished successfully in 1.100 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testAndroidTestCaseSetupProperly finished successfully in 0.500 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testDeleteSingleRecord finished successfully in 0.225 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testInsertAndSelect finished successfully in 0.275 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testPreconditions finished successfully in 0.150 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testQuery finished successfully in 0.200 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testUpdateSingleRecord finished successfully in 0.250 s.
* Class ch.bfh.kryptonIT.android.test.SecretInputActivityTest : Test testOrientation finished successfully in 1.626 s.
* Class ch.bfh.kryptonIT.android.test.SecretInputActivityTest : Test testPreconditions finished successfully in 0.825 s.
* Class ch.bfh.kryptonIT.android.test.RobotiumTest : Test testAddStore finished successfully in 22.152 s.
* Class ch.bfh.kryptonIT.android.test.RobotiumTest : Test testQueryStoreExtended finished successfully in 17.151 s.
* Class ch.bfh.kryptonIT.android.test.RobotiumTest : Test testQueryStoreSimple finished successfully in 10.100 s.

* Class ch.bfh.kryptonIT.android.test.InputlistActivityTest : Test testOrientation finished successfully in 1.725 s.
* Class ch.bfh.kryptonIT.android.test.InputlistActivityTest : Test testPreconditions finished successfully in 1.450 s.
* Class ch.bfh.kryptonIT.android.test.KeyInputActivityTest : Test testKeyList finished successfully in 1.476 s.
* Class ch.bfh.kryptonIT.android.test.KeyInputActivityTest : Test testOrientation finished successfully in 1.555 s.
* Class ch.bfh.kryptonIT.android.test.KeyInputActivityTest : Test testPreconditions finished successfully in 0.725 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testDefaults finished successfully in 1.250 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testDigitsSeekBar finished successfully in 1.100 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testFactorSeekBar finished successfully in 1.275 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testMissingName finished successfully in 1.325 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testOrderSeekBar finished successfully in 1.125 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testOrientationChangeAdvancedTab finished successfully in 1.350 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testOrientationChangeMainTab finished successfully in 1.300 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testPreconditions finished successfully in 1.251 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testSecurityHigh finished successfully in 1.225 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testSecurityInsane finished successfully in 1.375 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testSecurityNormal finished successfully in 1.225 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testWrongMinMaxPoints finished successfully in 1.425 s.
* Class ch.bfh.kryptonIT.android.test.InputFreeActivityTest : Test testOrientation finished successfully in 1.784 s.
* Class ch.bfh.kryptonIT.android.test.InputFreeActivityTest : Test testPreconditions finished successfully in 0.850 s.
* Class ch.bfh.kryptonIT.android.test.StoreListActivityTest : Test testOrientationChange finished successfully in 1.454 s.
* Class ch.bfh.kryptonIT.android.test.StoreListActivityTest : Test testPreconditions finished successfully in 0.825 s.
* Class ch.bfh.kryptonIT.android.test.StoreListActivityTest : Test testStoreList finished successfully in 1.000 s.

F.2.3. Fassade

Test results

08.12.2011

Summary:

Total 16 tests in 5 classes.
Total errors: 0
Total failures: 0
Tests succeded: 16
Total time: 0.256 s

Successes

* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testSecretJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testKappaJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testPolynomJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testSecretStoreJson finished successfully in 0.001 s.
* Class ch.bfh.kryptonIT.facade.test.TestDeflaterSize : Test testDeflatorSize finished successfully in 0.231 s.
* Class ch.bfh.kryptonIT.facade.test.TestKryptex : Test testNoRandomPointsStore finished successfully in 0.001 s.
* Class ch.bfh.kryptonIT.facade.test.TestKryptex : Test testSigmaBackAndForth finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestKryptex : Test testCreateAndRetrieveNumberStore finished successfully in 0.003 s.
* Class ch.bfh.kryptonIT.facade.test.TestKryptex : Test testCreateAndRetrieveTextStore finished successfully in 0.012 s.
* Class ch.bfh.kryptonIT.facade.test.TestZipedJson : Test testSecretJson finished successfully in 0.003 s.
* Class ch.bfh.kryptonIT.facade.test.TestZipedJson : Test testKappaJson finished successfully in 0.001 s.
* Class ch.bfh.kryptonIT.facade.test.TestZipedJson : Test testPolynomJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestZipedJson : Test testSecretStoreJson finished successfully in 0.003 s.
* Class ch.bfh.kryptonIT.facade.test.TestSecretConversion : Test testSEcretBaseConversion finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestSecretConversion : Test testSecretConversion finished successfully in 0.001 s.
* Class ch.bfh.kryptonIT.facade.test.TestSecretConversion : Test testSecretValidationRegex finished successfully in 0.000 s.

F.3. 3. Iteration

Hier sind die Resultate der im Kapitel E definierten Tests enthalten.

F.3.1. Web Applikation

Durchgeführt von Louis Bernath am 06.01.2012 für Iteration 3.

Testfall	Ergebnis
Applikation aufrufen	Ok
Tabwechsel nach Store erstellen	Ok
Ansicht wechseln	Ok
Finiter Körper erstellen	Ok
Schlüsseleingabe erster Schlüssel	Ok
Schlüsseleingabe zweiter Schlüssel	Ok
Schlüsseleingabe abschliessen	Ok
Geheimniseingabe	Ok
Geheimniseingabe 2	Ok
Polynominterpolation	Ok
Polynominterpolation 2	Ok
Store speichern	Ok
Store speichern 2	Ok
Tabwechsel nach Store laden	Ok
Ansicht wechseln	Ok
Store laden	Ok
Geheimnis abfragen	Ok
Geheimnis abfragen 2	Ok
Tabwechsel nach Import/Export	Ok
Store exportieren	Ok
Store importieren	Ok
Store exportieren 2	Ok
Store importieren 2	Ok
Store löschen	Ok

JUnit Testresultate

Test results

04.01.2012

Summary:

Total 13 tests in 1 classes.
Total errors: 0
Total failures: 0
Tests succeded: 13
Total time: 12.352 s

Successes

* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testSimple finished successfully in 8.665 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testFiniteBodyWidgetDefaultValuesSet finished successfully in 1.651 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testKeysWidget finished successfully in 0.151 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testKeysWidgetWithThreeKeys finished successfully in 0.193 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testKeysWidgetWithTwoKeysWithTwoKeyParts finished successfully in 0.171 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testSecretsWidget finished successfully in 0.060 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testSecretsWidgetWithThreeSecrets finished successfully in 0.062 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testPolynomInterpolationWidget finished successfully in 0.064 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testSaveStoreWidget finished successfully in 0.120 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testRetrieveSecretWidget finished successfully in 0.090 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testRetrieveSecretWidgetWithTwoKeyParts finished successfully in 0.105 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testImportWidget finished successfully in 0.546 s.
* Class ch.bfh.kryptonIT.gwt.client.WidgetTests : Test testExportWidgetDefaultValueSet finished successfully in 0.474 s.

Test results

04.01.2012

Summary:

Total 32 tests in 1 classes.
Total errors: 0
Total failures: 0
Tests succeded: 32
Total time: 76.221 s

Successes

* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSimple finished successfully in 9.202 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testFiniteBodyValidatorSecretTypeEmpty finished successfully in 3.886 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testKeysValidatorFiniteBodyEmpty finished successfully in 2.118 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testKeysValidatorKeyPartEmpty finished successfully in 2.086 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testKeysValidatorKeyPartsIdentic finished successfully in 2.096 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSecretsValidatorFiniteBodyEmpty finished successfully in 2.052 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSecretsValidatorKeysEmpty finished successfully in 2.053 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSecretsValidatorSecretEmpty finished successfully in 2.052 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSecretsValidatorOnlyNumbers finished successfully in 2.047 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSecretsValidatorOnlySigns finished successfully in 2.079 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSecretsValidatorTooLong finished successfully in 2.063 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorFiniteBodyEmpty finished successfully in 2.078 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorFiniteBodyEmpty2 finished successfully in 2.066 s.

* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorKeysEmpty finished successfully in 2.067 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorKeysEmpty2 finished successfully in 2.075 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorSecretsEmpty finished successfully in 2.048 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorSecretsEmpty2 finished successfully in 2.053 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorRandomPointsEmpty finished successfully in 2.064 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorRandomPointFieldEmpty finished successfully in 2.068 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testPolynomInterpolationValidatorRandomPointFieldNotNumber finished successfully in 2.048 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSaveStoreValidatorStoreEmpty finished successfully in 2.082 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSaveStoreValidatorStoreNameEmpty finished successfully in 2.066 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testSaveStoreValidatorStoreNameInvalid finished successfully in 2.060 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testLoadStoreValidatorNoStoreSelected finished successfully in 2.064 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testRetrieveSecretValidatorNoStoreLoaded finished successfully in 2.079 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testRetrieveSecretValidatorKeyPartEmpty finished successfully in 2.086 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testRetrieveSecretValidatorKeyPartsIdentic finished successfully in 2.106 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testImportExportValidatorImportEmpty finished successfully in 2.357 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testImportExportValidatorImportInvalid finished successfully in 2.262 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testImportExportValidatorExportNoStoreSelected finished successfully in 2.283 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testImportExportValidatorExportNoExportTypeSelected finished successfully in 2.240 s.
* Class ch.bfh.kryptonIT.gwt.client.ValidatorTests : Test testImportExportValidatorDeleteNoStoreSelected finished successfully in 2.235 s.

Test results

04.01.2012

Summary:

Total 15 tests in 1 classes.
Total errors: 0
Total failures: 0
Tests succeded: 15
Total time: 14.487 s

Successes

* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testSimple finished successfully in 12.956 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetSpaceCreateDataAsync finished successfully in 0.557 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetKappaDataAsync finished successfully in 0.103 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetSigmaToYAsync finished successfully in 0.063 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetKappa2Async finished successfully in 0.069 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetYAsync finished successfully in 0.082 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetSigmaToSecretAsync finished successfully in 0.069 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetJsonStringAsync finished successfully in 0.074 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testCompressedJsonStringAsync finished successfully in 0.068 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetCombinedKeyAsync finished successfully in 0.052 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testIsValidTextSecretAsync finished successfully in 0.063 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetConvertedSecretListAsync finished successfully in 0.104 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetDeconvertedSecretAsync finished successfully in 0.052 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetStoreFromJSonStringAsync finished successfully in 0.084 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITAsyncTests : Test testGetStoreAsync finished successfully in 0.088 s.

Test results

04.01.2012

Summary:

Total 5 tests in 1 classes.
Total errors: 0
Total failures: 0
Tests succeded: 5
Total time: 19.110 s

Successes

* Class ch.bfh.kryptonIT.gwt.client.KryptonITRPCTests : Test testSimple finished successfully in 10.122 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITRPCTests : Test testKeysWidget_withRPC finished successfully in 2.681 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITRPCTests : Test testSecretsWidget_withRPC finished successfully in 2.091 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITRPCTests : Test testPolynomInterpolationWidget_withRPC finished successfully in 2.101 s.
* Class ch.bfh.kryptonIT.gwt.client.KryptonITRPCTests : Test testRetrieveSecretWidget_withRPC finished successfully in 2.115 s.

F.3.2. Mobile Applikation

Durchgeführt von Jan Liechti am 06.01.2012 für Iteration 3.

Testfall	Ergebnis
Applikation starten	Ok
Neuer Store erstellen	Ok
Store Parameter eingeben	Ok
Schlüssel hinzufügen	Ok
Teil-Schlüssel hinzufügen	Ok
Zweiten Schlüssel hinzufügen	Ok
Schlüssel Eingabe abschliessen	Ok
Geheimnis eingeben	Ok
Zweites Geheimnis eingeben	Ok
Store Erstellung abschliessen	Ok
Store auswählen	Ok
Schlüssel hinzufügen	Ok
Schlüssel Eingabe abschliessen	Ok
Store Abfrage abschliessen	Ok

JUnit Testresultate

Test results

03.01.2012

Summary:

Total 37 tests in 9 classes.
Total errors: 0
Total failures: 0
Tests succeded: 37
Total time: 385.417 s

Successes

* Class ch.bfh.kryptonIT.android.test.SecretRetrieveActivityTest : Test testOrientation finished successfully in 6.725 s.
* Class ch.bfh.kryptonIT.android.test.SecretRetrieveActivityTest : Test testPreconditions finished successfully in 6.826 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testAndroidTestCaseSetupProperly finished successfully in 79.129 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testDeleteMultiple finished successfully in 0.650 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testDeleteSingleStore finished successfully in 0.700 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testInsertAndSelect finished successfully in 1.375 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testPreconditions finished successfully in 0.625 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testQuery finished successfully in 0.725 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testUpdateMultiple finished successfully in 0.650 s.
* Class ch.bfh.kryptonIT.android.test.provider.SecretStoreProviderTest : Test testUpdateSingleStore finished successfully in 1.350 s.
* Class ch.bfh.kryptonIT.android.test.SecretInputActivityTest : Test testOrientation finished successfully in 25.851 s.
* Class ch.bfh.kryptonIT.android.test.SecretInputActivityTest : Test testPreconditions finished successfully in 14.351 s.
* Class ch.bfh.kryptonIT.android.test.RobotiumTest : Test testAddStore finished successfully in 64.405 s.

* Class ch.bfh.kryptonIT.android.test.RobotiumTest : Test testQueryStoreExtended finished successfully in 39.252 s.
* Class ch.bfh.kryptonIT.android.test.RobotiumTest : Test testQueryStoreSimple finished successfully in 25.802 s.
* Class ch.bfh.kryptonIT.android.test.InputlistActivityTest : Test testOrientation finished successfully in 3.375 s.
* Class ch.bfh.kryptonIT.android.test.InputlistActivityTest : Test testPreconditions finished successfully in 3.575 s.
* Class ch.bfh.kryptonIT.android.test.KeyInputActivityTest : Test testKeyList finished successfully in 2.000 s.
* Class ch.bfh.kryptonIT.android.test.KeyInputActivityTest : Test testOrientation finished successfully in 1.900 s.
* Class ch.bfh.kryptonIT.android.test.KeyInputActivityTest : Test testPreconditions finished successfully in 2.375 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testDefaults finished successfully in 4.826 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testDigitsSeekBar finished successfully in 4.875 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testFactorSeekBar finished successfully in 4.950 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testMissingName finished successfully in 4.926 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testOrderSeekBar finished successfully in 5.156 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testOrientationChangeAdvancedTab finished successfully in 12.451 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testOrientationChangeMainTab finished successfully in 8.725 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testPreconditions finished successfully in 6.600 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testSecurityHigh finished successfully in 5.151 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testSecurityInsane finished successfully in 15.403 s.
* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testSecurityNormal finished successfully in 5.150 s.

* Class ch.bfh.kryptonIT.android.test.StoreParametersActivityTest : Test testWrongMinMaxPoints finished successfully in 11.927 s.
* Class ch.bfh.kryptonIT.android.test.InputFreeActivityTest : Test testOrientation finished successfully in 2.761 s.
* Class ch.bfh.kryptonIT.android.test.InputFreeActivityTest : Test testPreconditions finished successfully in 2.050 s.
* Class ch.bfh.kryptonIT.android.test.StoreListActivityTest : Test testOrientationChange finished successfully in 3.150 s.
* Class ch.bfh.kryptonIT.android.test.StoreListActivityTest : Test testPreconditions finished successfully in 3.125 s.
* Class ch.bfh.kryptonIT.android.test.StoreListActivityTest : Test testStoreList finished successfully in 2.550 s.

F.3.3. Fassade

Test results

04.01.2012

Summary:

Total 17 tests in 5 classes.
Total errors: 0
Total failures: 0
Tests succeded: 17
Total time: 7.729 s

Successes

* Class ch.bfh.kryptonIT.facade.test.TestStoreManager : Test testNoRandomPointsStore finished successfully in 0.289 s.
* Class ch.bfh.kryptonIT.facade.test.TestStoreManager : Test testSigmaBackAndForth finished successfully in 0.002 s.
* Class ch.bfh.kryptonIT.facade.test.TestStoreManager : Test testCreateAndRetrieveNumberStore finished successfully in 4.361 s.
* Class ch.bfh.kryptonIT.facade.test.TestStoreManager : Test testCreateAndRetrieveTextStore finished successfully in 0.050 s.
* Class ch.bfh.kryptonIT.facade.test.TestStoreManager : Test testCombiningKeys finished successfully in 0.004 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testSecretJson finished successfully in 0.016 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testKappaJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testPolynomJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestJson : Test testSecretStoreJson finished successfully in 0.003 s.
* Class ch.bfh.kryptonIT.facade.test.TestDeflaterSize : Test testDeflatorSize finished successfully in 0.052 s.
* Class ch.bfh.kryptonIT.facade.test.TestZipedJson : Test testSecretJson finished successfully in 2.948 s.
* Class ch.bfh.kryptonIT.facade.test.TestZipedJson : Test testKappaJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestZipedJson : Test testPolynomJson finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestZipedJson : Test testSecretStoreJson finished successfully in 0.004 s.
* Class ch.bfh.kryptonIT.facade.test.TestSecretConversion : Test testSecretBaseConversion finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestSecretConversion : Test testSecretConversion finished successfully in 0.000 s.
* Class ch.bfh.kryptonIT.facade.test.TestSecretConversion : Test testSecretValidationRegex finished successfully in 0.000 s.

G. Aufgabenstellung

für	Jan Liechti Louis Bernath
Fachbereich	Informatik
Betreuung durch	Reto Koenig Dr. Bernhard Anrig

KryptonIT

Ausgangslage
Die Entdeckung einer neuen kryptographischen Komponente (Secret-Storage Scheme) für das unabhängige sichere Speichern mehrerer Geheimnisse in einem Chiffrat (Secret-Store) führte zum Studentenprojekt KryptonIT. In diesem Projekt haben sich die Studierenden mit der zu diesem Zeitpunkt noch nicht veröffentlichten Komponente auseinandergesetzt und deren Eigenschaften studiert, beschrieben und ansatzweise implementiert. Aufbauend auf diesen Erkenntnissen und Erfahrungen soll nun eine Bachelorarbeit daraus hervorgehen, welche einerseits dem Interessierten die grundsätzlichen Fähigkeiten der Komponente in interaktiver Weise erläutern und andererseits eine Applikation zur Verfügung stellen soll, welche für erweiterte Brauchbarkeitsstudien herangezogen werden kann.

Aufgabe
Diese Arbeit befasst sich mit der Konkretisierung des Secret-Storage Schemes. Es sollen zwei Sichten des Secret-Storage Schemes erarbeitet werden:

- Erklärende Sicht:
  Ziel dieser Darstellung ist es, einen beschreibenden Zugang zum Secret-Storage Scheme zu schaffen. Dabei soll das Hauptaugenmerk auf dem Verständnis der Funktionsweise und Sicherheit eines konkreten Secret-Storage Systems liegen. Die zu verwendende Technologie für die interaktive Darstellung soll plattformunabhängig und möglichst transparent für den Benutzer sein. Dazu bietet sich die reine Webtechnologie mit ihren neuen interaktiven Elementen an, welche innerhalb dieser Arbeit Verwendung finden sollen.
- Benutzer Sicht:
  Ziel dieser Darstellung ist es, einen benutzerorientierten Zugang zum Secret-Storage Scheme zu schaffen. Dabei soll das Hauptaugenmerk auf der intuitiven Bedienung des Ablaufs für das Erstellen sowie das Abrufen von Secret-Stores liegen. Zusätzlich zur klassischen Pin-Eingabe ("What I know") für das Erstellen eines Passworts, soll auch der Kontext als mögliche Quelle angedacht und exemplarisch zur Verfügung gestellt werden ("What I have"). Da es keine einheitliche Art der Visualisierung kontextbezogener Daten gibt, wird der Lösung dieses Problems grosses Gewicht beigemessen. Als Technologieplattform bietet sich das Mobile-Phone an, welches mit Hilfe seiner vielen Sensoren sehr gut als Lieferant kontextbezogener Daten dienen kann.

Beide Sichten müssen die Möglichkeit haben, einen Secret-Store zu ex- und importieren. Dies ermöglicht den Vergleich der Resultate der beiden Sichten und kann zur Verifikation der Funktionalität herangezogen werden. Da die Forschungsarbeiten aber nicht auf den Aspekt der konkreten Speicherung eines Secret-Stores eingehen, ist für diesen Punkt noch erheblicher Engineering-Aufwand in zukünftigen Arbeiten zu treiben.

Während der Projektarbeit ist eine konkrete Basisrealisierung eines Secret-Storage Systems

in der Programmiersprache Java entstanden. Um diese Realisierung direkt in die Bachelorarbeit einfliessen lassen zu können, bieten sich Google-Webtoolkit als Plattform für die Webtechnologie sowie Googles Android als mobile Plattform an.

Beginn der Arbeit	19. September 2011
Abschluss der Arbeit	20. Januar 2012

Der Betreuer: Der Fachbereichsleiter:

H. Arbeitsjournal

Woche 18: Freitag, 20.01.2012

Abgabe der Arbeit

Woche 18: Donnerstag, 19.01.2012

- Dokumentation (Jan Liechti und Louis Bernath)
  - Letztes Review und Korrekturen
  - Zusammenstellen der Lieferergebnisse
  - Drucken der Dokumente

Woche 18: Mittwoch, 18.01.2012

- Dokumentation (Jan Liechti und Louis Bernath)
  - Review und Korrekturen

Woche 18: Dienstag, 17.01.2012

- Dokumentation (Jan Liechti und Louis Bernath)
  - Gesamter Bericht durchgelesen und überarbeitet.
  - GWT Benutzerdokumentation durchgelesen und überarbeitet.
  - Seiten-Umbrüche kontrolliert.
  - Bibliographie und Glossar korrigiert.
  - Bild-Beschreibungen korrigiert.
  - Titel-Seite für "kleinere" Dokumente korrigiert
  - Einleitung von Projekt-Handbuch und Anforderungen neu erstellt.
  - GANT Projekt-Plan neu generiert.

Woche 18: Montag, 16.01.2012

- Dokumentation: Überarbeitung aufgrund Feedbacks von R. König, B. Anrig und R. Bach (Louis Bernath)
- Benutzerdokumentation GWT fertig erstellt. (Jan Liechti)
- Mit Review des Berichts begonnen. (Jan Liechti)

Woche 17: Sonntag, 14.01.2012

- Dokumentation
  - Android Benutzerdokumentation fertig gestellt. (Louis Bernath)
  - Review und Korrekturen des Bericht-Hauptteils. (Louis Bernath)
  - Referenzen eingefügt. (Jan Liechti)
  - diverse Kapiteldetails in den Anhang verschoben. (Jan Liechti)
  - GWT Benutzerdokumentation weiter erstellt. (Jan Liechti)
  - Design Kapitel angepasst. (Jan Liechti)
  - Kapitel Ausblick bearbeitet. (Jan Liechti)
  - Kapitel Iteration 3 bearbeitet. (Jan Liechti)
  - Korrekturen des Bericht-Hauptteils. (Jan Liechti)
- Book (Jan Liechti und Louis Bernath)
  - Lead erstellt
  - Inhalte aus dem Management Summary übernommen
  - Screenschot von Android und GWT Applikation hochgeladen

Woche 17: Samstag, 14.01.2012

- Dokumentation Android (Louis Bernath)
  - Überarbeitung Zielplattform -> Mobile
  - Erstellung Analyse NFC und QR-Codes
  - Anpassung Glossar (ohne Nummerierung)
  - Korrektur Journal Wochen-Zahlen
  - Umsetzung Android: QR-Codes mit ZXing und Überarbeitungen
  - Ausblick: Ausformulierung der Stichworte.
  - Android Klassendiagramm neu generiert
  - Anhang "Iterationen": Android Iteration 3
  - Anpassung "Ausblick"
  - Projektplan angepasst (Meilensteine "Book", "Verteidigung", "Poster", "Präsentation")
  - Benutzerdokumentation: erste Inhalte (Installation, Store Erstellung und Entwicklung)

- Dokumentation GWT Umsetzung Kapitel (Jan Liechti)
  - Kapitel Graphische Ausgabe bearbeitet.
  - Kapitel Zufallspunkte bearbeitet.
  - Kapitel Darstellung der Achsenbeschriftung bearbeitet.
  - Kapitel Umrechnung bearbeitet
  - Kapitel Export erstellt.
  - Kapitel Info Bereich erstellt.
  - Kapitel Presenter erstellt.
  - Paketdiagramm erstellt und dokumentiert.
  - Klassendiagramm erstellt.
  - Kapitel Views erstellt.
  - Kapitel Events bearbeitet.
  - Grafik für Eventklassen erstellt.

Woche 17: Freitag, 13.01.2012

- Dokumentation (Jan Liechti und Louis Bernath)
  - Journal aus den Blog-Einträgen erstellt.
  - Überarbeitungen und Verbesserungen (Management Summary, Einführung, Analyse, Design, Schlussfolgerungen).
  - Generieren der GWT UI Design Bilder.
  - Seiten-Weise inkludieren der PDF Test-Resultate
  - Seiten-Weise inkludieren des PDF Journals
  - Kapitel MVP, GWT JUnit Tests im GWT Analyseteil erstellt.
  - Kapitel Store speichern und laden, Store importieren, Store exportieren in GWT Umsetzung erstellt.

Woche 17: Donnerstag, 12.01.2012

- Dokumentation (Jan Liechti)
  - Ergänzungen: Schlussfolgerung und Ausblick
  - Umsetzung: Javascript deaktiviert, Import Export, Dialog Boxen für Hilfetexte
  - Kapitel 3. Iteration mit Stichworten ergänzt

Woche 17: Mittwoch, 11.01.2012

- Dokumentation (Louis Bernath)
  - Ergänzungen: Schlussfolgerung und Ausblick
  - Analyse und Umsetzung: Hash-Methoden und modulare Exponentiation
  - Umsetzung: Hilfe und Activities

Woche 17: Dienstag 10.01.2012

- LaTeX (Louis Bernath)
  - Bericht ist nun zweiseitig und hat korrekte Kopfzeilen
  - Aufgabenstellung eingebunden.
- Dokumentation (Jan Liechti)
  - Management Summary ergänzt
  - Schlussfolgerung und Ausblick ergänzt
  - GWT
    - Import/Export im Kapitel Analyse ergänzt.

Woche 17: Montag 09.01.2012

Treffen mit Betreuern:

- Verifizieren der Deliverables.
- Diskussion Inhalt für Benutzerdokumentation (für End-Benutzer)

GWT: (Jan Liechti)

- DialogBoxen mit den Platzhaltern der Help-Anzeigen fertig erstellt und mit Lorem-Ipsum-Texten abgefüllt

Woche 16: Sonntag 08.01.2012

- Dokumentation (Louis Bernath)
  - Erste Inhalte für Management Summary
  - Erste Inhalte für Schlussfolgerung
  - Android Umsetzung: komplettes Klassendiagramm
- GWT (Jan Liechti)
  - JavaDoc fertig erstellt.
- Android (Louis Bernath)
  - Englische Ressourcen erstellt

Woche 16: Samstag 07.01.2012

- Dokumentation Ergänzungen und Überarbeitungen (Louis Bernath)
  - Analyse: Fassade und Mobile
  - Design: Mobile
  - Umsetzung: Fassade und Mobile
- Benutzerdokumentation initialisiert (Louis Bernath)
- GWT (Jan Liechti)
  - JavaDoc zu 85% erstellt

Woche 16: Freitag 06.01.2012

- Tests: 3. Iteration abgeschlossen (Jan Liechti und Louis Bernath)
- Dokumentation (Jan Liechti und Louis Bernath)
  - UseCases überarbeitet
  - Testresultate
  - Klassendiagramme für Android erstellt
- GWT Dokumentation (Jan Liechti)
  - Import und Export Kapitel im Analyse Teil erstellt

Woche 16: Donnerstag 05.01.2012

- Dokumentation GWT (Jan Liechti)
  - Kapitel Internationalisierung erstellt.
  - Kapitel Loading erstellt.
  - RPC Kapitel überarbeitet.
  - Web-Analyse und Web-Implementations Kapitel neu gegliedert und fehlende Titel erstellt.
- Dokumentation Android (Louis Bernath)
  - Analyse: Kompass-Eingabe
  - Umsetzung: Eingabe-Typen

Woche 16: Mittwoch 04.01.2012

- Dokumentation (Louis Bernath)
  - JUnit Test-Resultate für Android und die Fassade für die letzte Iteration ergänzt.
  - Überarbeitung der Test-Fälle für Android
  - Ergänzen der Test-Fälle für die Fassade.
  - Kapitel Web Applikation in Grundlagen erstellt und erster Entwurf geschrieben.
- Dokumentation (Jan Liechti)
  - Kapitel Google Web Toolkit in Analyse erstellt und erster Entwurf geschrieben.
  - GWT: Kapitel Validierungen erstellt und alle Validierungen dokumentiert.
  - GWT: Informationsanzeigen aufgelistet und dokumentiert.
  - GWT: Loading Seite dokumentiert.
  - GWT: Layout-Aufbau dokumentiert.

Woche 16: Montag 02.01.2012, Dienstag 03.01.2012

- Android: (Louis Bernath)
  - Validierung für duplizierte Teilschlüssel und Schlüssel eingefügt.
  - javadoc abgeschlossen
  - Code Cleanup & Bug Fixing
  - Fehler- und Informations-Dialoge ergänzt
- Fassade: (Louis Bernath)
  - javadoc abgeschlossen
  - Code Cleanup
- GWT (Jan Liechti)
  - Zusammentragen von Notizen
  - Analyse was gibt es bei der Dokumentation noch alles zu tun
  - JavaDoc der GWT JUnit Tests erstellt.
- Dokumentation (Jan Liechti)
  - GWT JUnit Test Kapitel erstellt.
  - Testfälle GWT für die 3. Iteration angepasst.
  - Testprotokoll für die 3. Iteration erstellt.

Woche 15: Sonntag 01.01.2012

- GWT (Jan Liechti)
  - GWT JUnit Tests überarbeitet.
  - weitere GWT JUnit Tests erstellt.
  - GWT JUnit Tests abgeschlossen.

Woche 15: Freitag 30.12.2011, Samstag 31.12.2011

- GWT (Jan Liechti)
  - 28 Validierungen eingebaut.
  - 31 JUnit Tests für die Validierungen erstellt.
  - Autoswitch des StackLayoutPanel beim Wechsel der Schritte nur bei der Benutzer Sicht eingeführt.
  - Fehler DropDown Aktualisierungen behoben.
  - Validierungen schön formatiert.
  - InfoDialogBox statische Breite, roter Text bei Fehlermeldungen, grüner Text bei Informationen.
  - Design überarbeitet.
  - 95% aller Texte internationalisiert.

Woche 15: Donnerstag 29.12.2011

- Dokumentation (Jan Liechti)
  - Abschluss der Umstrukturierung des Inhalts
  - Kapitel Einführung, Problemstellung angepasst
  - Kapitel Grundlagen erster Entwurf abgeschlossen

Woche 15: Mittwoch 28.12.2011

- Android (Louis Bernath)
  - Schlüssel mit Präfixe
  - Kompass Eingabe
  - Untersuchung der NFC Eingabe.
  - Store Parameter mit AsyncTask

- Facade (Louis Bernath)
  - Unterstützung für Schlüssel mit Präfixe

- KryptonIT (Louis Bernath)
  - Neue HashMethode EXP gemäss Reto König ergänzt/umgesetzt.

- GWT (Jan Liechti)
  - Knöpfe während der Durchführung der jeweiligen Methode disablen
  - Fehler X-Achse untersucht --> Problem lag an den verwendeten Hash-Methoden
  - Fehler ein/ausblenden behoben
  - Fehler Secret Ausgabe behoben
  - Darstellung des Infobereichs verschönert
  - Ausgabe Polynom verschönert
  - Anzeige SecretSpaceData hinzugefügt
  - Zusätzliche Infoboxen erstellt
  - Begrüssungsseite erstellt
  - Verzögerung der Schritte nur bei gewissen Anzeigen durchführen
  - Diverse Layout und Style Anpassungen durchgeführt
  - neue HashMethode EXP eingefügt

Woche 15: Dienstag 27.12.2011

- Restrukturierung und neu Gliedern der Dokumentation (Louis Bernath)

Woche 14: Freitag, 23.12.2011

- Dokumentation (Jan Liechti und Louis Bernath)
  - Einarbeitung des Feedbacks von René Bach

- Besprechung mit Reto (Jan Liechti und Louis Bernath)
  - Dokument-Struktur
  - Key Kombination
  - Integration Analog Compass

- GWT (Jan Liechti)
  - Infoboxen für Hilfetexte bearbeitet
  - Platzhalter bei Infoboxen eingefügt

Woche 14: Donnerstag 22.12.2011

- Android (Louis Bernath)
  - Analyse Compass & NFC
  - Rudimentäre Einbindung von "Analog Compass"

- GWT (Jan Liechti)
  - Grafische Ausgabe verschönert
  - Kappa und Sigma mit Trennlinie in grafischer Ausgabe eingefügt
  - Problem beim Wechseln der Ansichten behoben
  - Problem bei Geheimnisausgabe behoben

Woche 14: Dienstag, 20.12.2011

- GWT (Jan Liechti)
  - Diverse Layout-Anpassungen/Verschönerungen durchgeführt.
  - Weniger Beschriftungen bei den Achsen.
  - Scientific Darstellung angepasst.
  - Beschriftung im Zeichnungsbereich ist neu rechtsbündig.

Woche 14: Montag 19.12.2011

Treffen mit René Bach (Jan Liechti und Louis Bernath)

  - Input's entgegengenommen bezüglich Gliederung und Struktur des Berichts.
  - Input's entgegengenommen bezüglich Inhaltlicher Mängel.
  - Entscheid: Lernkomponente ist nicht Teil der Thesis.

Woche 13: Samstag 17.12.2011, Sonntag 18.12.2011

- Android (Louis Bernath)
  - Neue Import / Export Funktionalität und UI
    - Neu Import / Export mit der Android Zwischenablage
    - Neu Import / Export mit Android Dateisystem (.json Dateien)
    - Angepasstes UI mit QR Import / Export.
  - Anzeige der Store Informationen
  - Erweitertes Hilfe-System (4 Seiten)
  - Angepasstes UI für Wifi-Eingabe

- GWT (Jan Liechti)
  - Im Zeichnungsbereich wurden die Mouseover, welche mit SVG erstellt wurden, durch PopupPanels von GWT ersetzt. So werden die Popups auch über den Zeichnungsbereich hinaus angezeigt.
  - Rest vom Wochenende krank!!!

Woche 13: Freitag 16.12.2011

- GWT (Jan Liechti)
  - Export als Barcode umgesetzt
  - Loading beim Laden der Seite eingebaut
  - begonnen mit dem Umschreiben aller Horizontal- und VerticalPanel in FlowPanel. -> so werden keine Tabellen sondern nur div's generiert!

- Android (Louis Bernath)
  - Manelle Eingabe Wifi umgesetzt

- Besprechung Key Kombination (Jan Liechti und Louis Bernath)
- Import/Export zwischen Android und Webapplikation umgesetzt und getestet (Jan Liechti und Louis Bernath)

Woche 13: Dienstag 13.12.2011, Mittwoch 14.12.2011

- GWT (Jan Liechti)
  - Importieren/Exportieren eines Stores als JsonString umgesetzt
  - Löschen der im Browser gespeicherten Stores eingebaut
- Android (Louis Bernath)
  - Einfaches Hilfe-System (1 Seite)
  - Anpassungen der UI Texte

Woche 13: Montag 12.12.2011

- Dokumentation
  - Überarbeiten und Ergänzen der bestehenden Dokumentation (Jan Liechti)
  - Erkenntnisse GWT 2. Iteration erster Entwurf erstellt (Jan Liechti)
  - Erkenntnisse Android 2. Iteration erster Entwurf erstellt (Louis Bernath)

- Iteration 2 „Tag" erstellt (Louis Bernath)
- Dokumentations-Release erstellt und an René Bach gesendet (Louis Bernath)

Woche 12: Samstag 10.12.2011, Sonntag 11.12.2011

- Dokumentation
  - Kapitel Analyse, Implementation und Anhang angepasst und ergänzt (Jan Liechti)
  - Kapitel Analyse, Fassade ergänzt und überarbeitet (Louis Bernath)
  - Kapitel Analyse, Android UI Design überarbeitet (Louis Bernath)
  - Kapitel Implementation, Fassade ergänzt und überarbeitet (Louis Bernath)
- GWT (Jan Liechti)
  - Untersuchung GWT Unit Tests. -> für local storage nicht möglich und für Seekbar mit JQuery auch nicht möglich

Woche 12: Freitag 09.12.2011

- Android (Louis Bernath)
  - Bug mit Orientation Change geflickt

- GWT (Jan Liechti)
  - Anpassung auf aktuelle Fassade durchgeführt

- Dokumentation und Diskussion (Jan Liechti und Louis Bernath)
  - Diskussion mit Reto
  - Testfälle für Iteration 2 ergänzt
  - Überarbeitung und Ergänzung Kapitel 2 KryptonIT Methode
  - Generierung Junit Testprotokolle an Jan erklärt
  - Iteration 2 abgeschlossen

Woche 12: Mittwoch 07.12.2011, Donnerstag 08.12.2011

- Android (Louis Bernath)
  - Kleinere Fehler behoben
  - Unit Tests
    - erweitert mit "Orientation Change" (wechseln des Bildschirms von Hoch zu Quer-Format, oder umgekehrt).
    - erweitert mit Store Parameter Tests
    - an 2. Iteration angepasst

- Dokumentation (Jan Liechti)
  - Kapitel 2 (Kryptographische Methode Krypton IT) erster Entwurf erstellt.
  - 2 Skizzen zu Kapitel 2 erstellt.
  - GWT Analyse Kapitel teilweise angepasst.

Woche 12: Dienstag 06.12.2011

- Fassade (Louis Bernath)
  - Entfernen des "Security Order" aus dem Secret Store (der Parameter wird nur bei der Erstellung verwendet, und ist dann implizit im Polynom vorhanden)
- Android (Louis Bernath)
  - Kleine Korrekturen (Schreibfehler in den Ressourcen)
  - Anpassung an die geänderte Fassade
- GWT (Jan Liechti)
  - Projekthandbuch-Anpassung der GWT Meilensteine
    - Release 2 auf 08.12. verlängert
    - Release 3 und 4 zusammengefügt

Woche 12: Montag 05.12.2011

Meeting mit Betreuer (Jan Liechti und Louis Bernath)

- Klärung offener Fragen
  - Security Order für Abfrage nicht nötig
  - "Leere" Key und Secret nicht möglich
  - Key Kombinations-Algorithmus
- Demonstration und Diskussion von Android und GWT Applikation
- Nächste Treffen 09.01.2012 @ 17:00

Woche 11: Samstag 03.12.2011, Sonntag 04.12.2011

- Android (Louis Bernath)
  - Ikonen für Eingabe-Typen, Geheimnisse, Schlüssel-Kombinationen und Teile
  - Geheimnis Typ "Text" umgesetzt
  - Neues UI für die Anzeige von Geheimnissen (Abfrage)
  - Neuer und vereinfachter Abfrage-"Screen Flow"
  - Text Resources aufgeräumt
  - Kleinere Verbesserung des Layouts (ListView Einträge)
  - Zustand Speicherung bei "Runtime Changes"

- GWT (Jan Liechti)
  - Ablauf der einzelnen Schritte verzögert einblenden umgesetzt.
  - Layout Anpassungen abgeschlossen (Informationen nach rechts)
  - weitere Texte internationalisiert
  - Konzept für Hilfetexte überall eingebaut.
  - Linien verbinden die einzelnen Punkte -> Ablauf

Woche 11: Freitag 02.12.2011

- Besprechung Ablage des Secret Types in Space-Objekt und anschliessende Anpassung (Jan Liechti und Louis Bernath)
- Android (Louis Bernath)
  - UI Umsetzung Key Eingabe mit kombinierten Keys
- GWT (Jan Liechti)
  - Die einzelnen Schritte beim Store abfragen werden nacheinander mit einer kurzen Verzögerung ausgeführt.
  - Konzept für Hilfetexte überlegt und begonnen mit der Umsetzung davon

Woche 11: Donnerstag 01.12.2011

- GWT (Jan Liechti)
  - Kombination von Schlüsseln eingebaut.
  - Konzept der 5 Sichten (5 Angriffsmodelle) umgesetzt.

Woche 11: Mittwoch 30.11.2011

- Android (Louis Bernath)
  - Umsetzung des Store Parameter UI
  - Umsetzung des Key Input UI (mit Key Kombinationen)
- GWT (Jan Liechti)
  - Layout Anpassungen
    - Knöpfe und Ansichten reduziert
    - Ausgaben nach rechts verlagert
  - Validierungen eingebaut

Woche 11: Montag 28.11.2011

- GWT (Jan Liechti)
  - Texteingabe bei Secret eingebaut

Woche 10: Samstag 26.11.2011, Sonntag 27.11.2011

- Android (Louis Bernath)
  - Anpassung und Erweiterung Dokumentation: UI Design
  - Umsetzung UI "Store Parameter"

- GWT (Jan Liechti)
  - Anpassung Layout
    - Space Ansicht angepasst
  - Slidebar Widget erstellt, da es von GWT standardmässig keine Slidebar (Seekbar) gibt. Es dauerte sehr lange (10 Std.) und viele unterschiedliche Versuche, bis endlich eine Möglichkeit gefunden wurde, welche auch mit GWT 2.4 funktioniert!
  - begonnen mit Trennung von Ausgabe und Eingabebereich
  - die meisten Texte internationalisiert

Woche 10: Freitag 25.11.2011

- Besprechung und Verarbeitung der Kommentare von René Bach zur Dokumentation (Jan Liechti und Louis Bernath)
- UI Analyse (Jan Liechti und Louis Bernath)
    - Wahl der Store Parameter
    - Überarbeitung der UI Layouts (2. Iteration)
- Fassade (Jan Liechti und Louis Bernath)
    - JSON Fehler beseitigt

Woche 10: Dienstag 22.11.2011, Mittwoch 23.11.2011

- GWT (Jan Liechti)
    - GWT-Unit Tests erstellt. (Wenn im zu testenden Code ein Asynchroner Aufruf gemacht wird muss mit einem Timer der weitere Verlauf verzögert werden!)
    - Konzept für Internationalisierung studiert
    - Konzept der Internationalisierung umgesetzt. (Nun müssen noch alle Texte über die Properties-Files eingebunden werden)
- Android (Louis Bernath)
    - Analyse QR Code
    - Test QR Code Austausch zwischen Android Geräten
    - WiFi Schlüssel Eingabe aus Test-Applikation übernommen
    - QR Code Schlüssel Eingabe umgesetzt.

Woche 10: Montag 21.11.2011

- Treffen mit Experten und Betreuer (Jan Liechti und Louis Bernath)
    - Demonstration der ersten Iteration GWT und Android.
    - Termin für Verteidigung auf den 26.01.2012 @ 17:30 festgelegt.
    - Termin für nächstes Treffen mit Betreuern auf den 05.12.2011 @ 17:00 festgelegt.
    - Besprechung Services in Android.
    - Besprechung "Sichten" in GWT.

- Refactoring/Erweiterung der JSON Serialisierung (Louis Bernath)
    - Stragety und Decorator Pattern angewendet.
    - Gezipte Serialisierung als neue "Strategie" und "Dekorierung" für String Serialisierung (JSON) eingeführt.
    - Unit Test's für Gezipte JSON Secret Stores.

Woche 9: Samstag 19.11.2011, Sonntag 20.11.2011

- Dokumentation (Louis Bernath)
  - Secret Space Analyse
  - Screen Flow Analyse
- Anpassung Projektplan: Ergebnisse des Android Release 2 (Louis Bernath)
- KryptonIT Fassade erweitert (Louis Bernath)
  - Schlüssel Kombinieren
  - Zahlenbasis Umrechnung
  - Mapping: Secret <-> Secret Text
- GWT (Jan Liechti)
  - MVP Pattern umgesetzt
  - Fehlermeldungen (wie z.B. keine Verbindung) werden nicht als Popup sondern in einer Dialogbox dargestellt.

Woche 9: Frietag 18.11.2011

- Gemeinsame Analyse (Jan Liechti und Louis Bernath)
  - GWT Model-View-Presenter
  - SecretSpace und die Wahl der möglichen Zeichen

- SVN-Tag für Iteration 1 erstellt. (Louis Bernath)

- GWT: Umsetzung MVP (Jan Liechti)
- Android: Dokumentation "Analyse" (Louis Bernath)

Woche 9: Mittwoch 16.11.2011, Donnerstag 17.11.2011

- GWT (Jan Liechti)
  - Untersuchungen Model-View-Presenter Pattern

Woche 9: Dienstag 15.11.2011

- Treffen mit R. Bach und Betreuer findet am 21.11.2011 @ 13:00 im MZ statt! (Louis Bernath)
- LaTeX (Louis Bernath)
  - Dokumentklasse von scrartcl (Artikel) auf scrreprt (Report) geändert (KOMA Skript)
  - Anpassung aller Gliederungen (section -> chapter, subsection -> section, etc.)
  - Anpassung des Inhaltsverzeichnis für Glossar, LOF, LOT und Bibliografie

- Iteration 1 abgeschlossen! (Jan Liechti und Louis Bernath)

Woche 8: Samstag 12.11.2011, Sonntag 13.11.2011

- Test (Jan Liechti und Louis Bernath)
  - Benutzertest der GWT Applikation
  - Android Unit-Test Methoden ergänzt.
  - Benutzertest der Android Applikation
- Dokumentation (Jan Liechti und Louis Bernath)
  - Testresultate und Feedback für die GWT Applikation.
  - Testresultate und Feedback für die Android Applikation.
  - Dokumentation der ersten Iteration im Kapitel "Umsetzung".
  - Dokumentation der Erkenntnisse für die erste Iteration.
  - Android Unit-Testresultate der ersten Iteration im Anhang eingefügt.

Woche 8: Freitag 11.11.2011

- Besprechung MVP Pattern für GWT (Jan Liechti und Louis Bernath)
- Inhalt Umsetzungsdokumentation abgesprochen (Jan Liechti und Louis Bernath)
- Testfälle definiert (Jan Liechti und Louis Bernath)

Woche 8: Mittwoch 09.11.2011, Donnerstag 10.11.2011

- GWT (Jan Liechti)
  - Rechteck als Hintergrund für Mouseover Texte erstellt.
  - Dokumentation Analyse Teil ergänzt (ev. können diverse Teile davon in die Umsetzung verschoben werden)
- Android (Louis Bernath)
  - Erster Robotium Blackbox Test: "Store erstellen"
  - Dokumentation "Umsetzung":
    - Erster Wurf, Text "Fassade"
    - Stichortartig Kapitel "Android"
    - Erster Wurf, Fazit 1. Iteration

Woche 8: Montag 07.11.2011

- GWT (Jan Liechti)
  - einige Funktionstests durchgeführt und ein paar kleine Fehler korrigiert.
  - Studie GWT mit local storage
  - Load Store und Save Store in/aus dem Browser (local storage) umgesetzt
  - Scroll Problematik gelöst
  - SVG texte verschönert (ohne Stroke)

Woche 7: Sonntag 06.11.2011

- Android (Louis Bernath)
  - Unit Test für Content Provider für "update" und "delete" hinzugefügt.
  - Text für das UI wird von Resourcen geladen.
  - JavaDoc für alle Klassen eingefügt.
  - Gradient-Farbe auf der SeekBar hinzugefügt.

- GWT (Jan Liechti)
  - zusätzliche Eventhandler für jeden Knopf eingebaut
  - Punkte setzen auf der Secret Achse
  - Punkte setzen auf der Key Achse
  - Beschriftung Key Achse
  - Beschriftung Achsenformat Scientific erstellt
  - Store anzeigen löschen (Create Store und Retrieve Store)
  - Koordinaten System für Retrieve Store
    - Achsenbeschriftung aller Achsen
    - Punkte auf Achsen eintragen
  - Texteingabe bei Schlüssel Eingabe (nicht nur Zahlen)
  - Mehrere Schlüssel-Geheimnispaare beim Erstellen eines Stores ablegen
  - Erste JUnit Tests erstellt

Woche 7: Samstag 05.11.2011

- GWT (Jan Liechti)
  - Zufallspunkte hinzufügen Funktion eingebaut.
  - Alle Punkte werden auf Click gesetzt
  - Punkte haben Beschriftung on mouseover
  - Beschriftung der Achsen dynamisiert
- Android (Louis Bernath)
  - Validierung der Benutzereingaben
    - Validierung, dass Store Name gesetzt ist.
    - Validierung, dass Secret Space gesetzt ist.
    - Grad muss vom Benutzer nicht >= 100 gewählt werden.
    - Warnung wenn keine Keys gesetzt werden.
    - Warnung wenn "leerer" (=0) Key verwendet wird.
    - Warnung wenn "leeres" (=0) Secret verwendet wird.
    - Validierung, dass Secret <= Secret Space.
  - Asynchrone erstellung des Secret Stores (AsyncTask).
  - Unit Test's für Activities und Content Provider erweitert/hinzugefügt.
  - Ikonen hinzugefügt.
- Fassade (Louis Bernath)
  - Mock/Fake Objekte hinzugefügt.
  - Methoden "equals" und "hashCode" implementiert.

Woche 7: Freitag 04.11.2011

- Gemeinsame Untersuchung bezüglich "custom GWT event handler" für Datenübergabe. (Jan Liechti und Louis Bernath)
- Gemeinsame Untersuchung bezüglich JUnit Tests mit asynchroner Callbacks. (Jan Liechti und Louis Bernath)
- Android: Studium und Umsetzung von Activity Unit Tests. (Louis Bernath)
- GWT: Methode für das Setzen von Punkten im Koordinatensystem. (Jan Liechti)

Woche 7: Dienstag 01.11.2011, Donnerstag 03.11.2011

- Android (Louis Bernath)
  - Erstellen und Abfragen eines Stores umgesetzt
  - Freie Eingabe
  - Laden und Speichern eines Stores mit ContentProvider (SQLite)
  - Store Parameter (security order, hash method)
- Fassade (Louis Bernath)
  - Polynom JSON de- und Serializierung angepasst
- GWT (Jan Liechti)
  - Studie: JUnit in GWT. --> funktioniert nun. :-)
  - Studie: Clientseitige globale Variablen in GWT

Woche 7: Montag 31.10.2011

- Review und Korrektur der Dokumentation für Veröffentlichung. (Jan Liechti und Louis Bernath)
- E-Mail mit Terminvorschlägen an Herrn Bach. (Louis Bernath)

Woche 6: Samstag 29.10.2011, Sonntag 30.10.2011

- Recherche JSON (Louis Bernath)
- Fassade mit JSON Serialisierung und Deserialisierung ausgestattet. (Louis Bernath)
- Android (Louis Bernath)
  - Erstellung eines Stores funktioniert rudimentär.
  - UI für die Erstellung eines Stores weiterentwickelt.
  - Weitergabe der Erstellungs-Parameter als Intent-Extras
  - Umsetzung Conent Provider mit SQLite Datenbank
  - Test's mit Android API Level 8 (anstatt des ursprünglich gewählten Level 7)
- Dokumentation (Louis Bernath)
  - Anpassung Projekthandbuch und Projektplan auf aktuellen Stand
  - Überarbeitung Bericht
  - Ergänzen des Glossars
- Recherche QR Code als Speicher für einen Store -> funktioniert nicht, da grösster QR Code nicht von Android gelesen werden kann und die Anderen zu klein sind (Louis Bernath)
- GWT Applikation (Jan Liechti)
  - Überarbeitung Aufbau des Zeichnungsteils
  - Erstellung von Widgets, welche alle Komponenten (Eingabefelder, Buttons) eines Erstellungsschrittes zusammenfasst
  - Recherche Cookies als Speicher für einen Store -> funktioniert nicht, da nur 4k gross
  - Layout verschönert
    - StackLayoutPanel für die Aufteilung der Schritte verwendet

    - Probleme mit JUnit Tests
        - bis jetzt funktioniert es noch nicht, JUnit Tests in GWT zu erstellen.

Woche 6: Freitag, 28.10.2011

- Arbeiten (Jan Liechti und Louis Bernath)
    - Android UI Umsetzung
    - GWT Widgets für die einzelnen Erstellungsschritte erstellt
    - GWT Refactoring des bestehenden Codes

- Treffen mit Betreuer (Jan Liechti und Louis Bernath)
    - Zukünftige Treffen am Freitag abgesagt.
    - Weitere Treffen voraussichtlich am Montag 16:00. Definitive Entscheidung fällt nach oder vor Treffen mit Experten.
    - Treffen mit Experten und Betreuer am 7.11 oder 14.11, Management Zentrum oder ISB, Mittag oder ~16:00
    - Missverständnis betreffend "PIN Eingabe" und "Freie Eingabe"
        - Anpassung Pflichtenheft
        - Anpassung Bericht
    - Besprechung GWT Layout Vorschlag mit Reto König
        - GWT Layout wird in der 2. Iteration angepasst
            - bei der Passwortabfrage gibt es weniger Unterteilungsschritte
            - die Ausgaben werden zentral an einem Ort platziert

Woche 6: Donnerstag, 27.10.2011

- Android UI Implementation (Louis Bernath)
- GWT Layout Grundaufbau (Jan Liechti)

Woche 6: Dienstag, 25.10.2011

- Dokumentation (Jan Liechti)
    - Kapitel Ausgangslage neu geschrieben
    - Kapitel Aufgabe neu geschrieben

Woche 6: Montag 24.10.2011

- Treffen mit Experten Herr Dr. Bach (Jan Liechti und Louis Bernath)
- Abmachungen
    - Treffen in ca. 3 Wochen (inkl. Betreuer)
    - Lieferung Dokumentation in ca. 2 Wochen (Vorbereitung für Treffen)

Woche 5: Samstag 22.10.2011, Sonntag 23.10.2011

- Abschluss Android Analysen (Louis Bernath)
  - Anpassung Wireframes und Screenflow
  - Visualisierung von WLAN Eingaben.

- Abschluss GWT Analysen (Jan Liechti)
  - Screenflow ist noch zu erstellen!
  - Kapitel Layout und Design erstellt
  - Kapitel Aufbau der Applikation ergänzt
  - alle anderen Unterkapitel der Analyse kurz überarbeitet

Woche 5: Freitag 21.10.2011

- Dokumentation Analyseteil weitergeführt (Jan Liechti und Louis Bernath)
- KryptonIT Fasade fertig erstellt (Louis Bernath)

Woche 5: Mittwoch 19.10.2011, Donnerstag 20.10.2011

- Dokumentation (Jan Liechti)
  - Analyse GWT kurz weitergeführt
- Analyse Zeichnen mit GWT (Jan Liechti)
  - Bibliothek gwt-Graphics getestet
  - erster Prototyp des Zeichnungsteils mit gwt-Graphics erstellt

Woche 5: Montag 17.10.2011, Dienstag 18.10.2011

- Dokumentation der Android Anaylsen (Louis Bernath)
  - Wahl des Secret Space
  - UI Design
- Termin mit René Bach am 24.10.2011 @ 16:00 im ISB (Friedheimweg) bestätigt. (Louis Bernath)

- GWT (Jan Liechti)
  - GWT Layout Skizzen von Visio in Pencil übernommen
  - GWT Layout 1. Version fertig erstellt (Store erstellen und Store abfragen)
  - fortsetzen Dokumentation Analyseteil
  - GWT Aufbau Client-Server Kommunikation Skizze erstellt
  - begonnen mit Skizze Aufbau und Zusammenhang der einzelnen Pakete

Woche 4: Samstag 15.10.2011, Sonntag 16.10.2011

- Grundgerüst von der GWT Applikation erstellt (Jan Liechti)
- Dokumentation (Jan Liechti)
  - Anpassung GWT Applikationslayout (Jan Liechti)
  - GWT Analyse-Teil mit Bildern und Stichworten ergänzt (Jan Liechti)
  - Android Analyse: Systemkomponenten und Sicherheit (Louis Bernath)

Entscheidung: Analyse Phase wird bis zum 23.10.2011 verlängert! (Jan Liechti und Louis Bernath)

Woche 4: Freitag 14.10.2011

- Treffen mit Betreuer (Jan Liechti und Louis Bernath)
  - Besprechung UI Wireframes
  - Besprechung GWT Architektur (Serverseitige Verarbeitung)
- Kontaktaufnahme mit dem Experten Dr. René Bach (Jan Liechti und Louis Bernath)
- Android Wireframe Screen erstellt (Louis Bernath)
- Erste Version einer Fassade für die KryptonIT Bibliothek erstellt (Jan Liechti und Louis Bernath)
- Analyse Objektübergabe Server-Client in GWT (Jan Liechti und Louis Bernath)

Woche 4: Donnerstag 13.10.2011

- Android Komponenten einsetzen: UML Komponenten Diagramm erstellt (Louis Bernath)
- Android UI Wireframe's erstellt (Louis Bernath)
- Untersuchung GWT Serverseitig (Jan Liechti)

Woche 4: Montag 10.10.2011, Dienstag 11.10.2011

- Untersuchung: UI Wireframe Tool "Pencil" http://pencil.evolus.vn/ (Louis Bernath)
- Untersuchung Zeichnen mit GWT (Jan Liechti)
  - Canans Element angeschaut
  - gwt-svg installiert und erste Versuche damit durchgeführt

Woche 3: Samstag 08.10.2011, Sonntag 09.10.2011

- Übernahme der Grundlagen aus Proj2 (Louis Bernath)
- Dokumentieren der KryptonIT Parameter und Schritte. (Louis Bernath)
- Erstellen erster Android UI Wireframing. (Louis Bernath)
- Erweitern des Layouts von der GWT-Applikation (Jan Liechti)
- Untersuchung Zeichnen mit GWT (Jan Liechti)

Woche 3: Freitag 07.10.2011

- Finalisieren Pflichtenheft und Projekthandbuch (Jan Liechti und Louis Bernath)
- Analyse der User- und System-Parameter von der KryptonIT Library (Jan Liechti und Louis Bernath)
- Testklasse erstellt, welche die Schritte aufteilt (Jan Liechti und Louis Bernath)

Woche 3: Donnerstag 06.10.2011

- Dokumentieren der bereits durchgeführten Analysen und Tests mit Google Web Toolkit. (Jan Liechti)
- Requirements Dokument angepasst. (Jan Liechti und Louis Bernath)
- Android Screenflow, und Lösungskomponenten skizziert. (Louis Bernath)

Woche 3: Montag 03.10.2011, Dienstag 04.10.2011

- Erster grober unvollständiger Entwurf des Layouts von der GWT-Applikation erstellt. (Jan Liechti)
- Erste Analyse der KryptonIT Bibliothek, da dies für das Entwerfen des Layouts nötig war. (Jan Liechti)
- Aufbau des LaTeX Grundgerüsts für den Bericht. (Louis Bernath)
- Ergänzung des Pflichtenhefts. (Louis Bernath)

Woche 2: Samstag 01.10.2011

- Untersuchungen für KryptonIT clientseitig mit GWT 2.4 laufen zu lassen. (Jan Liechti)
    - Anfrage an dean.povey@gmail.com um Sachverhalt mit crypto-gwt zu klären.
- Erkenntnis: crypto-gwt läuft nicht client-seitig mit GWT 2.4.0 und App Engine 1.5.4. Laut der dem Maven-Manifest ist nur GWT 1.7.1 unterstützt. (Jan Liechti)
- Entscheidung: vorerst lassen wir die KryptonIT-Library serverseitig laufen. (Jan Liechti)

Woche 2: Freitag 30.09.2011

- Besprechung der Dokumente Pflichtenheft und Projekthandbuch mit Herrn Anrig und Reto . (Jan Liechti und Louis Bernath)
- Anpassungen und Ergänzungen des Besprochenen in den Dokumenten (Jan Liechti und Louis Bernath)
    - Dokument Versionen
    - Kapitel Qualitätsicherung (Projekthandbuch)
    - Kapitel Vorgehensmodell (Projekthandbuch)
    - Ergänzung Ausgangslage (Pflichtenheft)

- Festgestellung: KryptonIT mit GWT läuft nicht clientseitig. (Jan Liechti)

Woche 2: Mittwoch 28.09.2011, Donnerstag 29.09.2011

- Diverse Untersuchungen durchgeführt, um KryptonIT-Library mit GWT zu verwenden. (Jan Liechti)
    - Ergebnis: Mit den zusätzlichen Libraries crypto-gwt und gwt-async-future und einigen Anpassungen läuft nun die GWT Applikation mit der KryptonIT-Library
- GWT Zeichnungs-Tutorial durchgespielt (Jan Liechti)
- Studium der Junit-Tests in Android (Louis Bernath)
- Latex-Dokumente-Titelseite ausgelagert und in allen Dokumenten eingebunden (Louis Bernath)

Woche 2: Dienstag 27.09.2011

- Ergänzung der Resultate und Meilensteine. (Jan Liechti und Louis Bernath)
- Review und Korrektur der ersten Version. (Jan Liechti und Louis Bernath)
- Release der ersten Version: Pflichtenheft und Projektplan. (Louis Bernath)

Woche 2: Montag 26.09.2011

- Überarbeitung und Korrektur Projekthandbuch und Pflichtenheft. (Jan Liechti und Louis Bernath)
- Besprechung des aktuellen Standes und nötiger Arbeiten für einen ersten Entwurf des Projekthandbuches und Pflichtenhefts. (Jan Liechti und Louis Bernath)

Woche 1: Samstag 24.09, Sonntag 25.09

- Erweitern und ausführen der Anforderungen im Pflichtenheft. (Jan Liechti und Louis Bernath)
- Überarbeiten des Projekthandbuches. (Jan Liechti und Louis Bernath)
- Erfolglose Untersuchungen um KryptonIT-Library mit GWT zu verwenden (mit crypto-gwt). (Jan Liechti)

Woche 1: Freitag 23.09.2011

- Erste Version des Projekthandbuches erstellt (Jan Liechti und Louis Bernath)
  - Projektplan
  - Meilensteine und Phasen
  - Terminierung der Arbeiten
- Pflichtenheft Struktur erstellt (Jan Liechti und Louis Bernath)
  - Stichworte für das Pflichtenheft zusammengetragen
- Einbinden der Simplix Library in die Android-Umgebung (hat einwandfrei Funktioniert :-D) (Louis Bernath)

- Sehr viele Versuche durchgeführt, um die Library crypto-gwt in die GWT-Applikation einzubinden. Bisher leider erfolglos, da meiner Ansicht nach diese Library noch andere Pakete benötigt, welche aber nicht gefunden werden konnten. (Jan Liechti)

Woche 1: Erste Aktivitäten

Mittwoch: (Jan Liechti)

- Erste Versuche durchgeführt, die Library von Reto König einzubinden.
- Beim Kompilieren gab es jedoch noch Probleme mit den javax.crypto packages. (dieses Problem wird in den nächsten Tagen noch untersucht, ev. ist crypto-gwt die Lösung)

Donnerstag: (Louis Bernath)

- Begonnen mit einem stichwortartigen Entwurf des Projekthandbuches
- Ersten Entwurf des Zeitplans erstellt.
- Gedanken zum Pflichtenheft gemacht.

Woche 1: Verteilung erster Arbeiten

- Aufräumen und Strukturieren SVN für Thesis (Louis Bernath)
- Initial Import der KryptonIT Library ins SVN (Louis Bernath)
- Verteilung erster Arbeiten bis Freitag 23.09.2011: (Jan Liechti und Louis Bernath)
  - Individuelle Erarbeitung der Requirements (Pflichtenheft) und Planung Arbeits-Pakete/Schritte (Projekthandbuch)

Bachelor Thesis : KickOff

19.09.2011@20:15 Kick-Off mit Reto König und Bernhard Anrig. (Jan Liechti und Louis Bernath)

- Neuste Erkenntnisse von Reto König
  - Zusätzlicher "riesiger" Exponent -> mehr Sicherheit

- (Voraussichtliche) Meeting Termine für gesamte Bachelor Thesis abgemacht
  - 30.09.2011 @ 09:30-10:30
  - 14.10.2011 @ 09:30-10:30
  - 28.10.2011 @ 16:40-17:40
  - 11.11.2011 @ 09:30-10:30
  - 25.11.2011 @ 09:30-10:30
  - 09.12.2011 @ 09:30-10:30
  - 23.12.2011 @ 09:30-10:30
  - 13.01.2012 @ 09:30-10:30

- Entscheidungen
  - Ausgangslage für die Arbeit ist grundsätzlich das Paper "How to Store some Secrets" die Java-Library Stand vom 19.09.2011.
  - Meilensteine für erstes Meeting am 30.09.2011
    - Stabilität der KryptonIT-Library von Reto König sicherstellen.
    - Projektplan und Projekthandbuch erstellen.

Abbildungsverzeichnis

Literaturverzeichnis

[1] *Lock-icon*. http://commons.wikimedia.org/wiki/File:Lock-icon.svg. – By Hk kng (Own work) [GFDL (www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (www.creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

[2] KOENIG, Reto ; HAENNI, Rolf ; SPYCHER, Oliver ; SCHLÄPFER, Michael: *How to Store some Secrets*. befindet sich im Status Peer Review, 2011

[3] *KeePass Password Safe*. http://keepass.info/. – [Stand 21.03.2011]

[4] *Password Safe Simple & Secure Password Management*. http://passwordsafe.sourceforge.net/. – [Stand 21.03.2011]

[5] *GNU General Public License Version 2*. http://www.gnu.org/licenses/gpl.html. – [Stand 21.03.2011]

[6] *SplashID Secure Password Manager*. http://www.splashdata.com/splashid/. – [Stand 21.03.2011]

[7] *Artistic License 2.0*. http://www.opensource.org/licenses/artistic-license-2.0.php. – [Stand 21.03.2011]

[8] *Web Applikation*. http://de.wikipedia.org/wiki/Webanwendung. – Wikipedia Artikel über Web Applikation Definition, Eigenschaften, Vorteile, Nachteile, Sicherheit, Geschichte.

[9] *Rich Internet Application*. http://de.wikipedia.org/wiki/Rich_Internet_Application. – Rich Internet Application Definition, Eigenschaften, Technologien, Vorteile, Nachteile, Geschichte.

[10] *Ajax*. http://de.wikipedia.org/wiki/Ajax_(Programmierung). – Wikipedia Artikel über Ajax.

[11] *KeePass Password Generator*. http://keepass.info/help/base/pwgenerator.html. – KeePass Password Safe, unter GNU GENERAL PUBLIC LICENSE Version 2, Juni 1991

[12] LIECHTI, Jan ; BERNATH, Louis: *KryptonIT - Password Tresor*. 2011. – Projektarbeit für Modul „Projekt 2“, Stand 17. Juni 2011.

[13] *Modulare Exponentiation*. http://www.iti.fh-flensburg.de/lang/krypto/algo/modexp.htm. – FH Flensburg, Zahlentheoretische Algorithmen der Kryptographie

[14] *Google Web Toolkit*. http://de.wikipedia.org/wiki/Google_Web_Toolkit. – Google Web Toolkit Definition, Asynchrone Kommunikation, Verwendung von JavaScript

[15] *Google Web Toolkit)*. http://code.google.com/webtoolkit/. – Projekt Site von Google Web

Toolkit.

[16] *JUnit*. http://de.wikipedia.org/wiki/JUnit. – Wikipedia Artikel über JUnit.

[17] *Java-Swing*. http://de.wikipedia.org/wiki/Swing_(Java). – Wikipedia Artikel über Java-Swing.

[18] *Apache Tomcat*. http://de.wikipedia.org/wiki/Apache_Tomcat. – Wikipedia Artikel über Apache Tomcat.

[19] *Model-View-Presenter*. http://de.wikipedia.org/wiki/Model_View_Presenter. – Wikipedia Artikel über das Model-View-Presenter Entwurfsmuster.

[20] *Model-View-Controller*. http://de.wikipedia.org/wiki/Model_View_Controller. – Wikipedia Artikel über das Model-View-Controller Entwurfsmuster.

[21] *Building Testable Applications with Google Web Toolkit*. http://blog.codiform.com/2011/01/building-testable-applications-with.html. – Bericht, wie man Testbare Applikationen mit GWT erstellt. Erwähnt MVP.

[22] *Canvas (HTML-Element)*. http://de.wikipedia.org/wiki/Canvas_(HTML-Element). – Wikipedia Artikel über Canvas (HTML-Element).

[23] *LIB-GWT-SVG*. http://www.vectomatic.org/lib-gwt-svg. – lib-gwt-svg Bibliothek von Vectomatic

[24] *LIB-GWT-SVG Samples*. http://www.vectomatic.org/gwt/lib-gwt-svg-samples/lib-gwt-svg-samples.html. – Samples von der lib-gwt-svg Bibliothek von Vectomatic.

[25] *gwt-graphics)*. http://code.google.com/p/gwt-graphics/wiki/Manual. – Projekt Website von gwt-graphics. A cross-browser vector graphics library for GWT

[26] *gwt-graphics examples)*. http://hene.virtuallypreinstalled.com/gwt-graphics-examples/#mousemove. – gwt-graphics examples.

[27] *HTTP-Cookie)*. http://de.wikipedia.org/wiki/HTTP-Cookie. – Wikipedia Artikel über HTTP-Cookies.

[28] *HTTP-Cookie)*. http://code.google.com/webtoolkit/doc/latest/DevGuideHtml5Storage.html. – Developer's Guide - Client-side Storage (Web Storage)

[29] *JSON*. http://json.org/. – JavaScript Object Notation

[30] *Creating a GWT Wrapper for the JQuery UI Slider)*. http://www.zackgrossbart.com/hackito/gwt-slider/. – Tutorial um in GwT den JQuery UI Slider einsetzen zu können.

[31] *Unit Testing GWT Applications with JUnit)*. http://code.google.com/webtoolkit/doc/latest/tutorial/JUnit.html. – Tutorial über JUnit Tests mit GWT.

[32] *Android Fundamentals*. http://developer.android.com/guide/topics/fundamentals.htmll. – Android SDK Dokumentation, Fundamentals

[33] *Android Internal Storage*. http://developer.android.com/guide/topics/data/data-storage.html#filesInternal. – Android SDK Dokumentation, Internal Data Storage

[34] *Android Intents*. http://developer.android.com/guide/topics/intents/intents-filters.html. – Android SDK Dokumentation, Intents

[35] *Android Security*. http://developer.android.com/guide/topics/security/security.html. – Android SDK Dokumentation, Security

[36] *Android Storage*. http://developer.android.com/guide/topics/data/data-storage.html. – Android SDK Dokumentation, Data Storage

[37] *SQLite Features*. http://www.sqlite.org/features.html. – SQLite Projekt Website, Features

[38] *Android Responsiveness*. http://developer.android.com/guide/practices/design/responsiveness.html. – Android SDK Dokumentation, Responsiveness

[39] *Android Processes and Threads*. http://developer.android.com/guide/topics/fundamentals/processes-and-threads.htm. – Android SDK Dokumentation, Processes and Threads

[40] *Android Tasks and Back-Stack*. http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html. – Android SDK Dokumentation,Tasks and Back-Stack

[41] *Android Activity State*. http://developer.android.com/guide/topics/fundamentals/activities.html#SavingActivityStatel. – Android SDK Dokumentation, Activity State

[42] *Android Runtime Changes*. http://developer.android.com/guide/topics/resources/runtime-changes.html. – Android SDK Dokumentation, Runtime Changes

[43] *Android Resources*. http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources. – Android SDK Dokumentation, Thema Resources

[44] *Android Localization*. http://developer.android.com/guide/topics/resources/localization.html. – Android SDK Dokumentation, Localization

[45] *Android Testing*. http://developer.android.com/guide/topics/testing/testing_android.html. – Android SDK Dokumentation, Testing

[46] *Robotium*. http://code.google.com/p/robotium/. – Robotium, Projekt WEbiste, Android Blackbox Testing

[47] *Stackoverflow, Applikations-Spezifisches Keyboard*.
`http://stackoverflow.com/questions/1896939/android-app-specific-soft-keyboard.` – Forum-Eintrag über Android Applikations-Spezifisches Keyboard

[48] *Android Keyboard*.
`http://developer.android.com/reference/android/inputmethodservice/Keyboard.html.` – Android SDK Dokumentation, Thema Keyboards

[49] *QR-Code*. `http://de.wikipedia.org/wiki/QR-Code.` – Wikipedia Artikel über QR-Code.

[50] *ZXing Barcode Reader*. `http://code.google.com/p/zxing/.` – Projekt Website, ZXing Barcode Reader for Android

[51] *Near Field Communication*. `http://de.wikipedia.org/wiki/Near_Field_Communication.` – Wikipedia Artikel über NFC Merkmale, Anwendungsfälle, Marktprognosen

[52] *Near Field Communication*. `http://developer.android.com/guide/topics/nfc/index.html.` – Android SDK Dokumentation über Near Field Communication.

[53] *JSON in Java*. `http://json.org/java/.` – Java Klassenbibliothek für die Unterstützung von JSON.

[54] *Diskreter Logarithmus*. `http://de.wikipedia.org/wiki/Diskreter_Logarithmus.` – Wikipedia, Diskreter Logarithmus, Algorithmen zur Berechnung des diskreten Logarithmus

[55] *Making Remote Procedure Calls*.
`http://code.google.com/webtoolkit/doc/latest/tutorial/RPC.html.` – Beispiel eines Remote Procedure Calls mit GWT.

[56] *Developer's Guide - Layout Using Panels*.
`http://code.google.com/webtoolkit/doc/latest/DevGuideUiPanels.html.` – Wie man Layouts mit Panels in GWT erstellt.

[57] *Tab Layout Panel*.
`http://gwt.google.com/samples/Showcase/Showcase.html#!CwTabLayoutPanel.` – GWT Showcase - Tab Layout Panel.

[58] *Stack Layout Panel*.
`http://gwt.google.com/samples/Showcase/Showcase.html#!CwStackLayoutPanel.` – GWT Showcase - Stack Layout Panel.

[59] *Google Chart Tools: Infographics*.
`http://code.google.com/apis/chart/infographics/docs/qr_codes.html.` – QR-Code mit Google Chart Tools erstellen

[60] *Internationalizing a GWT Application*.
`http://code.google.com/webtoolkit/doc/latest/tutorial/i18n.html.` – Ein GWT Anwendung internationalisieren.

[61] *Android Analog Compass*. `http://code.google.com/p/android-ac/.` – Projekt Website, Android Analog Compass

[62] *Base64*. http://de.wikipedia.org/wiki/Base64. – Wikipedia Artikel über Base64 Kodierung.

[63] *Polynomialinterpolation - Newtonscher Algorithmus*. http://de.wikipedia.org/wiki/Polynominterpolation\#Newtonscher_Algorithmus. – [Stand 26.03.2011]

[64] PAAR, Christof ; PELZL, Jan: *Understanding Cryptography*. Springer-Verlag Berlin Heidelberg, 2010. – ISBN 978–3–642–04100–6

[65] *Hashfunktionen*. http://de.wikipedia.org/wiki/Hash-Funktion. – [Stand 26.03.2011]

[66] *crypto-gwt*. http://code.google.com/p/crypto-gwt/downloads/list. – crypto-gwt - Crypto library with JCE emulation for Google Web Toolkit.

[67] *gwt-async-future*. http://code.google.com/p/gwt-async-future/downloads/list. – gwt-async-future - Alternative approach to GWT AsyncCallback using functional programming techniques.

[68] *gwt-crypto*. http://code.google.com/p/gwt-crypto/downloads/list. – gwt-crypto - Crypto module for the Google Web Toolkit (GWT).

[69] *Scalable Vector Graphics*. http://de.wikipedia.org/wiki/Scalable_Vector_Graphics. – Wikipedia Artikell über SVG.