Kryptographie Praktikum: SSL Notaries

Technische Analyse und Design

EINFÜHRUNG

SSL Notaries stellen Dienste dar, welche als dritte Partei neben dem Client und der CA selbst die Korrektheit und Gültigkeit eines Zertifikats bezeugen sollen. Hierbei existieren mehrere Dienste, deren Interfaces im Folgenden kurz beschrieben werden sollen. Die Beschreibung der Interfaces soll als Grundlage für die Implementierung innerhalb des Kryptographie Praktikums dienen und eine zentrale Anlaufstelle für technische Fragestellungen darstellen. Weiterhin sollen Gemeinsamkeiten der Ansätze herausgearbeitet werden, um innerhalb des Praktikums den Code möglichst einfach und wiederverwendbar zu halten.

Weiterhin werden die Libraries genannt und kurz beschrieben, welche im Rahmen dieses Projekts eingesetzt werden sollen. Die Auswahl der Libraries erfolgt auf Basis der Beschreibungen der SSL Notaries. Es wird versucht, die Abhängigkeiten so gering wie möglich zu halten, sodass eine einfache Nutzung der Lösung ermöglicht wird.

Im letzten Kapitel soll das Design der Applikation anhand der Anforderungen abgeleitet und kurz skizziert werden. Hierzu erfolgt im ersten Teil die Nennung der Anforderungen und deren Auswirkungen und Abhängigkeiten zueinander. Im zweiten Teil wird auf Basis der Anforderungen ein Design entworfen und mittels UML visualisiert.

SSL NOTARY IMPLEMENTIERUNGEN

Die ICSI Certificate Notary

Die ICSI Certificate Notary sammelt passiv Informationen über vorhandene Zertifikate aus dem „upstream Traffic“ mehrere Internetkontenpunkte und speichert diese in einer zentralen Datenbank. Die Abfrage von Zertifikatsinformationen erfolgt über die DNS-Server des Projekts, wobei die angefragte Domain folgenden Aufbau haben muss:

```
<sha1>.notary.icsi.berkeley.edu
```

`<sha1>` stellt hierbei den Hashwert des Zertifikats dar, welcher durch den Browser oder durch die hier entwickelte Applikation vom zu testenden Zertifikat berechnet werden muss:

```
checkNotary(sha1(cert))
```

Je nach abgefragtem Resource Record existieren unterschiedliche Rückgabewerte:

1. Existiert die abgefragte Domain nicht in der ICSI Datenbank, so wird `NXDOMAIN` zurückgegeben. Diese Antwort ist unabhängig vom abgefragten Resource Record.
2. Existiert die abgefragte Domain in der ICSI-DB und wird ein A-RR abgefragt, dann existieren zwei Rückgabewerte:
   A. Der Rückgabewert `127.0.0.1` gibt an, dass das Zertifikat in der Datenbank vorhanden ist

B. Der Rückgabewert `127.0.0.2` gibt an, dass das Zertifikat innerhalb des hierarchischen Vertrauensmodells auf ein RootCA-Zertifikat aus dem „Mozilla root store“ zurückgeführt werden kann.

3. Existiert die abgefragte Domain in der ICSI-DB und wird ein TXT-RR abgefragt, dann enthält die Antwort fünf Attribute, denen mittels dem „=“-Zeichen Integer-Werte zugewiesen:

   - `version` Version der Antwort, Format
   - `first_seen` Unix Timestamp, zu dem das Zertifikat zum ersten Mal gesehen wurde
   - `last_seen` Unix Timestamp, zu dem das Zertifikat das letzte Mal gesehen wurde
   - `times_seen` Anzahl an Tagen zwischen first_seen und last_seen, an dem das Zertifikat mindestens einmal gesehen wurde.
   - `validated` Boolean Wert, welcher den Validierungsstatus des Zertifikats angibt. Der Zeitpunkt der Validierung ist nicht bekannt!
   - Beispiel:

     ```
     version=1 first_seen=15387 last_seen=15646 times_seen=260 validated=1
     ```

Es besteht Kompatibilität zu Google Certificate Catalogue. Hierbei werden die Versions- und Validated-Attribute unterdrückt und die Attributsnamen inkl. der „=“-Zeichen entfernt.

Mit dem Ansatz und den Rückgabewerten existieren mehrere Probleme:

1. Selbst-signierte Zertifikate oder Zertifikate unbekannter CAs können nicht validiert werden und enthalten daher immer `validated=0`, unabhängig ihrer Gültigkeit.

2. Es findet keine Überprüfung des Sperr-Status statt. Daher kann man trotz `validated=1` nicht sagen, ob das Zertifikat gesperrt wurde oder noch gültig ist.

3. Es wird keine Prüfung der Zuordnung zwischen Host und Zertifikat durchgeführt oder in der Datenbank gespeichert. Um ein Zertifikat bekannt zu machen und später als möglicherweise legitim darzustellen kann somit beispielsweise ein Testserver, welcher mit dem Angreiferzertifikat ausgestattet ist, einmal täglich aufgerufen werden. Zum Zeitpunkt des Angriffs ist das Zertifikat bereits lange bekannt, sodass eine Unterscheidung zum korrekten Zertifikat möglicherweise nicht mehr möglich ist.

4. Die im TXT-RR gespeicherten Werte können nur in Verbindung zueinander betrachtet werden und besitzen für sich selbst keine Aussagekraft. times_seen muss hierbei immer in Verbindung mit first_seen und last_seen gesehen werden, first_seen darf nicht zu weit in der Vergangenheit und last_seen muss möglichst nahe am aktuellen Datum liegen. Weiterhin muss der Abstand zwischen first_seen und last_seen groß genug sein, um eine Aussage darüber treffen zu können, ob das Zertifikat regelmäßig im Netzwerkverkehr gesehen wird.

5. first_seen bzw. last_seen haben keine Beziehung zur eigentlichen Gültigkeit des Zertifikats. first_seen kann daher zwar einen „optimalen“ Zeitpunkt des ersten Auftretens (z.B. vor einem Jahr) enthalten, jedoch kann das Zertifikat zu diesem Zeitpunkt bereits 20 Jahre gültig sein.

Die Berechnung des Scores erfolgt mittels folgendem Pseudocode. Als höchster Wert kann hierbei 50 erreicht werden, als niedrigster 0.

```
result = 0;
if(NXDOMAIN) return result;

// Test 1: A-RR

if(ip = 127.0.0.1) result += 5;
else if(ip = 127.0.0.2) result += 10;

// Test 2: TXT-RR

// a) times_seen should be near last_seen-first_seen
max_seen = last_seen - first_seen
min_seen = 0
// [min_seen,max_seen] -> [0,10]
result_a = 10*times_seen/max_seen

// b) last_seen should be near today: „near“ ~ [today-30,today]
// If result_b == 0 -> Certificate has not been seen for a long time
// and should be treated as possibly invalid.
max_seen = today
min_seen = today - 30days
// [min_seen,max_seen] -> [0,10]
result_b = 10*(last_seen - min_seen)/(max_seen - min_seen)
if(result_b<0) result_b = 0;

// c) first_seen should not be too far away from today
//    .. but also not too near..: Maybe use best practice from
// certificate validity periods: 2 years
// If result_c == 0 -> Certificate is very old
// and could already be compromised.
max_seen = today
min_seen = today - 2 years
optimal_seen = today - 1 year
if(first_seen == optimal_seen) result += 10;
if(first_seen > optimal_seen) result += 10 - 10*(x - optimal_seen)
                                          /(max_seen - optimal_seen)
if(first_seen < optimal_seen) result += 10*(x - min_seen)/(optimal_seen
```

```
                                                    - min_seen)

// d) first_seen and last_seen should not be too near to each other
// This is implicitly done in b) and c)

// e) If validated=1
if(validated=1) result_e = 10
else result = 0

// Calculate everything:
if(result_b > 0 && result_c > 0)
      result += result_a + result_b + result_c
result += result_e
```

Quellen:

- http://notary.icsi.berkeley.edu

Convergence

Die Grundlage von Convergence stellt das **Perspectives** Projekt dar, welches bereits als Browser Addon bereit steht, jedoch einige Nachteile aufweist, die im Rahmen von Convergence beseitigt wurden. Bei Perspectives wird - neben einem kontinuierlichen Monitoring von SSL-Sites - eine Verbindung sowohl vom Client, als auch von der Notary aufgebaut und die erhalten Zertifikate lediglich auf Clientseite verglichen. Der Notary wird das zuvor erhaltene Zertifikat dabei nicht mitgeteilt:

```
if(checkNotary(url)==“ok“)
      // Everything is fine for now…
```

Bei der Implementierung entstanden dadurch einige Probleme hinsichtlich

- Completeness: Das Browser-Addon untersucht nur die initiale Verbindung, jedoch nicht die innerhalb der Website aufgebauten Verbindungen, z.B. für Bilder, Stylesheets, usw…
- Privacy: Tracking von Zugriffen, wodurch Bewegungsprofile erstellt werden können.
- Responsiveness: Der sog. „Notary Lag“ bezeichnet die Zeit, in der ein Zertifikat von der Notary zwischengespeichert wird und dadurch die Antwort z.B. im Falle einer Zertifikatserneuerung, falsch sein kann.

Bei **Convergence** wurden unterschiedliche Lösungen für die o.g. Probleme entwickelt:

- Notary Lag: Dieses Problem wurde dadurch gelöst, dass das Zertifikat an die Anfrage an die Notar gebunden wird. Dadurch muss die Notary lediglich zum Server verbinden und erhält Match/Mismatch des Zertifikats und kann dies den Client mitteilen.

```
checkNotary(url, cert)
```

- Privacy: Local Caching:

```
if(!checkLocalCache(url, cert)      // Cache invalid or
                                    // cert not avail.
   && checkNotary(url, cert)==“ok“)
      addCache(url, cert);
```

- Privacy: Notary Bounce: Zentrale Notary, die weitere Notaries anfragen kann und dadurch die Zurückverfolgbarkeit eliminiert.
- Erweiterbarkeit: Über Convergence lassen sich alle weiteren Notaries prinzipiell anbinden und abfragen („Collective Trust“). Damit stellt Convergence genau das dar, was im Rahmen dieses Projekts entwickelt werden soll.

Es wird ein RESTful Protokoll implementiert, wobei JSON zur Repräsentation der Daten eingesetzt wird. Die Notaries sind auf `Port 443` und `4242` erreichbar. Port 80 kann weiterhin als HTTP Proxy für `CONNECT` Requests an `Port 4242` genutzt werden. Es sind zwei Anfragen möglich, die Antwort setzt sich aus JSON-String und HTTP-Response-Code zusammen:

1. Eine POST-Anfrage, welche den Fingerprint des Zertifikats enthält. Hierbei findet eine Überprüfung bereits auf auf der Convergence Notary statt.

```
POST /target/<host to validate>+<port>
fingerprint=<sha1(cert)>
```

Als Antwort kommen unterschiedliche HTTP-Status-Codes zum Einsatz.

- 200: Die Notary konnte das Zertifikat verifzieren. Im Body wird ein JSON String nach unten gezeigtem Aufbau übertragen.
- 409: Die Notary konnte das Zertifikat nicht verifizieren. Im Body wird ein JSON String nach unten gezeigtem Aufbau übertragen, welcher die (potentiell) korrekten FPs enthält.
- 303: Verifikation des FPs ist nicht möglich (z.B. wegen einer Eigenschaft, welche auf dem abgefragten Host nicht vorhanden ist — was könnte das sein????).
- 400: Fehler bei der Anfrage bzw. fehlender Parameter. Im Body wird eine lesbare Fehlermeldung mitgegeben.
- 503: Interner Fehler bei der Notary. Fehlermeldung im Body. Verifikation sollte als fehlgeschlagen angesehen werden.

2. Eine GET-Anfrage an die selbe URL. Hierbei wird lediglich die Liste der Fingerprints zurückgegeben, es findet jedoch keine Validierung auf der Notary statt.

```
GET /target/<host to validate>+<port>
```

Aus den Rückgabewerten der unterschiedlichen Notaries kann der Client einen Wert fürs „Collective Trust“ bestimmen. Hierbei können unterschiedliche Methoden zum Einsatz kommen.

[todo..]

Es existieren zwei Probleme bei diesem Ansatz, welche sich jedoch nur auf spezielle Umgebungen beziehen, in denen entweder keine Notary verfügbar ist oder welche abseits der Standards und Best Practices arbeiten.

- citybank problem: Different certificates for different connections
- captive portals: Hotel-WLAN: Kein Internet, keine Möglichkeit zum Testen

Quellen

- http://sarwiki.informatik.hu-berlin.de/Convergence
- https://github.com/moxie0/Convergence/wiki/Notary-Protocol

Crossbear

Crossbear ist ein auf Convergence aufbauendes Konzept der TU München zur Erkennung von MitM-Angriffen. Sowohl Server, als auch Clients sind über Github als OpenSource erhältlich. Die Datensammlung erfolgt analog zu Perspectives/Convergence über aktives Monitoring auf Basis von Client-Anfragen. Um weitere Aktionen durchzuführen, wurden sog. „Hunter“ entwickelt, welche Aufgaben bzgl. Erkennung übernehmen. Hierbei gibt es zwei Arten von Huntern:

- Add-on für Mozilla Firefox
- Eigener Dienst.

Hunter „jagen“ einen MitM und versuchen ihn mittels verschiedener Methoden zu identifizieren und zu lokalisieren. Diese Informationen werden zum zentralen Crossbear-Server gesendet und dort zur Analyse gespeichert:

1. Zertifikate der Verbindungen, welche entweder selbst hergestellt oder mitgeschnitten wurden. Dies ermöglicht die einfache Erkennung falscher Zertifikaten auf Basis bereits bekannter Zertifikate.
2. Traceroutes auf Basis von ICMP zum Ziel, um alle Hosts auf dem Weg zum Ziel in die Analyse einschliessen zu können. Dies ermöglicht die Lokalisierung eines möglichen Angreifers anhand der Schnittpunkt von „guten“ (es wurde ein gültiges Zertifikat geliefert) und „schlechten“ (es wurde ein falsches Zertifikat geliefert) Verbindungen.
3. Weitere Informationen werden über die Anbindung von auf Convergence basierenden Notaries gesammelt und zur Anreicherung der eigenen Daten genutzt, beispielsweise Häufigkeit oder last_seen-Werte (siehe Convergence-Kapitel).
4. Out-of-Band-Informationen: Genutzte CAs, WHOIS-Informationen, Geo-IP-Mapping

Zwischen Hunter/Client und Crossbear-Server wird immer eine `TLS`-Verbindung aufgebaut. In der aktuelle Implementierung ist das Zertifikat im Client hardcodiert und bei einem falschen Zertifikat wird die Verbindung automatisch unterbrochen.

Beim Verbindungsaufbau schickt der Client dem Server einen `CertVerifyRequest`, welcher die Zertitikatskette, den Hostnamen und die aufgelöste IP-Adresse enthält. Der Server speichert diese Anfragen zusammen mit dem Zeitpunkt der Anfrage und stellt ebenfalls eine Verbindung zum Server her, um das Zertifikat zur überprüfen. Gleichzeitig werden die Ergebnisse mit den o.g. Notaries überprüft und korreliert. Als Ergebnis sendet der Server eine `CertVerifiyResult` zurück an den Client. Optional kann diese Nachricht eine Anfrage an einen Traceroute enthalten, sofern der Verdacht überprüft werden soll, ob eine MitM vorliegt. Optional können ebenfalls eine `PublicIPNotification` und der `Timestamp` mitgesendet werden. Weiterhin erstellt der Crossbear-Server einen (Risk-)Score für das Zertifikat auf Basis der Eigenschaften des Zertifikats (Last Continuous Servation Persiod LCOP, Anzahl früherer Sichtungen, Rückgabewerte weiterer Convergence-Server.). Dieser wird dem Client ebenfalls mitgeteilt. Der Standardwert im Client, ab dem eine Warnung erfolgt, liegt bei 100.

Hunter holen sich vom zentralen Crossbear-Server Hunt-Aufträge ab oder bekommen diese in Form von `Traceroute`-Anfragen bei einer eigenen Anfrage zugewiesen. Der Hunt-Auftrag enthält den abzufragenden Host und eine ServerTime-Nachricht. Der Hunter baut eine Verbindung zum Host auf und speichert dabei sowohl `Traceroute`, als auch die Zertifikatskette und gibt dies an den Crossbear-Server zurück.s
Der Hunter muss einen `PublicIPRequest` durchführen.

Quellen:
- https://pki.net.in.tum.de/
- https://github.com/crossbear/Crossbear/tree/master/server/fourhundredfourtythree/src/crossbear

Weitere SSL Notaries und mögliche Module

1. MonkeySphere http://web.monkeysphere.info/why/#index1h2
2. DNSSEC/DANE (keine SSL Notary)
3. WoT-Anbieter: GNUnet

BESCHREIBUNG DER IMPLEMENTIERUNG

Architektur

Die Applikation wird getreu den Anforderungen in einer Client/Server-Architektur realisiert. Dadurch teilt sich die Applikation in drei Teile:

1. Den Core, welcher die Logik zur Anbindung der Notaries und der Berechnung eines Scores enthält. Der Core steht lediglich dem Server zur Verfügung. Die Steuerung erfolgt mit Hilfe einer Konfigurationsdatei.

2. Den Client, welcher ein Zertifikat auf seine Gültigkeit hin untersuchen will. Der Client teilt dem Server hier mit, welche URL er überprüft haben möchte, welches SSL Zertifikat er möglicherweise bereits erhalten hat und welche Notaries zu dieser Überprüfung genutzt werden sollen. Die Anfrage wird weiterhin noch mit einer Signatur über die abgefragten Informationen und das Ergebnis vor Veränderung geschützt.

3. Der Server, welcher den Core anbindet und nach außen eine Schnittstelle zur Abfrage von Zertifikaten anbietet. Der Server kann hierbei beliebige Interfaces nach außen bereit stellen und so potentiell beliebige Clients anbinden. Das Ergebnis einer Anfrage besteht dabei aus einem `float`-Wert und einer Signatur über die abgefragten Informationen und das Resultat.

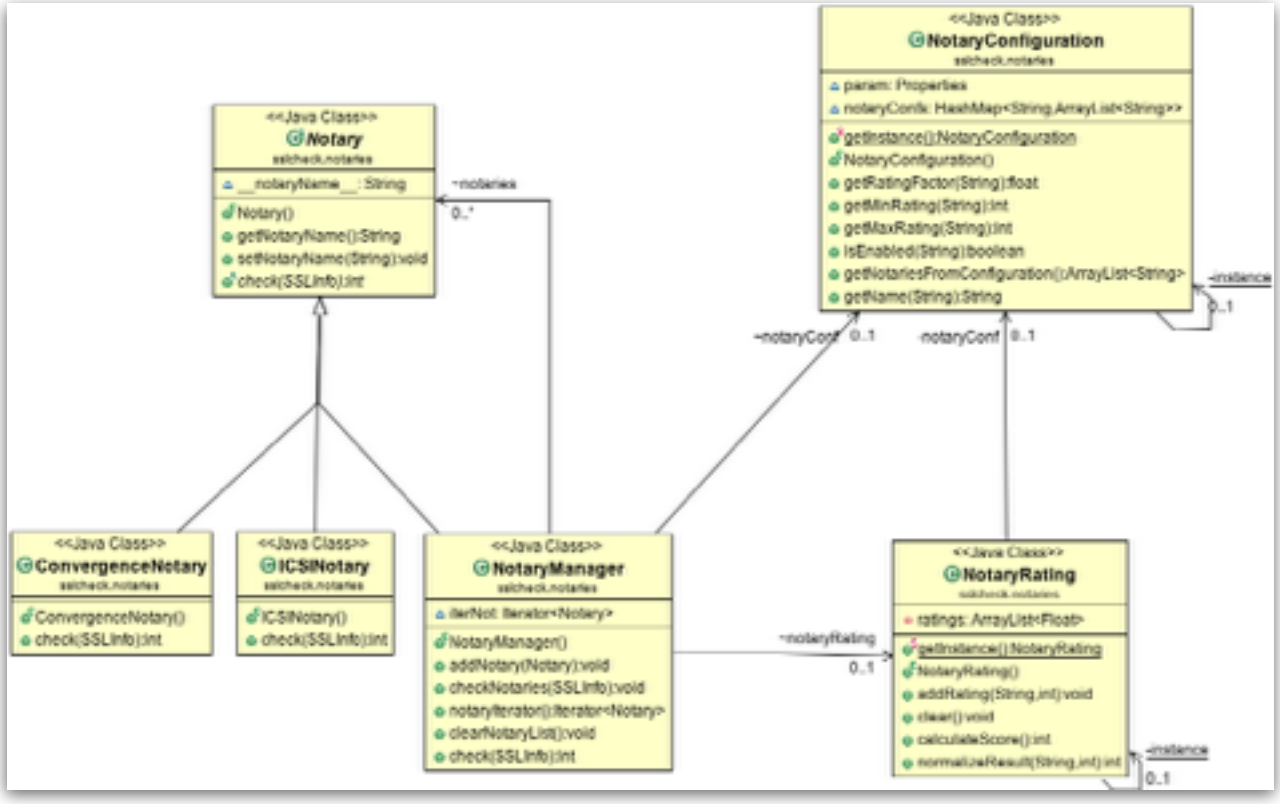
Erweiterbarkeit um neue Notaries spielt eine große Rolle. Aus diesem Grund wurde besonderen Wert auf den Core gelegt, welcher im folgenden beschrieben werden soll.

Core-Architektur

Der Kern der Applikation wird durch den NotaryManager gebildet. Dieser bildet durch ein abgewandeltes Proxy-Entwurfsmuster eine Abstraktionsebene zu den konkreten Notaries und steuert den Zugriff auf diese. Ebenfalls wird durch ihn die Komposition der Ergebnisse und die Berechnung eines Endergebnisses mit Hilfe der NotaryRating-Klasse angestoßen.

Unterstützt wird der NotaryManager durch die Singletons NotaryConfiguration und NotaryRating, welche die Konfiguration bzw. die Berechnungslogik der Bewertung bereitstellen. Sie werden vom NotaryManager zur Anbindung der Notaries und Verarbeitung derer Ergebnisse benötigt.

Die Implementation der Notaries erfolgt für jede Notary in ihrer eigenen Klasse. Für den Aufruf der Methode check() ist der NotaryManager verantwortlich. Weitere Methoden werden nicht aufgerufen. Der Server greift wie bereits beschrieben hierbei direkt auf die Notaries zu, sondern überlässt diese Aufgabe dem NotaryManager, wodurch eine einfache Erweiterbarkeit erreicht wird.



Diese Architektur ermöglicht dem Entwickler des Server eine einfache Anbindung von lediglich einer Notary. Alles weitere wird durch den NotaryManager übernommen. Dennoch können durch Entwickler leicht weitere Notaries hinzugefügt werden, in dem die Logik in einer Klasse implementiert wird und die Eigenschaften in der zentralen Konfigurationsdatei hinzugefügt werden. Eine Änderung am bestehenden Code ist hierbei nicht nötig.

Server-Architektur

Der Server baut auf dem Core auf und soll dessen Funktionalität einfach dem Client zur Verfügung stellen können. Hierzu bindet eine Serverimplementierung die Klasse sslcheck.core.NotaryManager an, da diese die volle Funktionalität zur Verfügung stellt und hierbei als Adapter zum Zugriff auf die Notaries, die Konfiguration und die Berechnungen dient.

Server/Client Beispiel: RESTServer/RESTClient

Als Beispielimplementation des Servers wird ein RESTful-Interface implementiert, welches mittels JSON angesprochen werden kann.

LIBRARIES UND TOOLS FÜR DIE REALISIERUNG

Technologien

- ICSI Notary: DNS
- Convergence: RESTful Webservices, JSON
- Crossbear: RESTful Webservices, JSON
- Interface zur Abfrage von Zertifikatsinformationen: RESTful Webservices, JSON

Libraries

- DNS: dnsjava (http://www.dnsjava.org/)
- RESTful Webservices: …
- Logging: log4j (https://logging.apache.org/log4j/2.x/)

Tools

- Buildtool: Maven3 (https://maven.apache.org/ref/3.0/) [Anleitung: todo]
- Entwicklungsumgebung: Eclipse (https://www.eclipse.org/)

ANFORDERUNGEN AN DAS DESIGN

Nicht-funktionale Anforderungen

- Leichte Erweiterbarkeit, modularer Aufbau

- Programmiersprache: Java

Funktionale Anforderungen

- Abfrage von Zertifikatsgültigkeitsinformationen mittels unterschiedlicher Interfaces
- Server/Client-Design, Client only Console Application
- Auswahl der abzufragenden SSL Notaries und Module beim Aufruf
- Rückgabe einer Bewertung der Antworten und Berechnung eines End-Scores

SPEZIFIKATION DER ANGEBOTENEN SCHNITTSTELLEN

Auswahl der Design Patterns

- Anbindung einzelner Module zur Abfrage der Notaries: Proxy, Factory Method
- Abfrage aller Module und Rückgabe der Werte: Iterator
- Anbindung Core<>Server: Adapter, Singleton

Skizzierung des Serverdesigns

AUSBLICK

Probleme

Das Problem, dass selbst die Abfrage des Dienstes angegriffen wird und die Antworten möglicherweise gefälscht wird, wurde hierbei außer Acht gelassen.

Steigerung der Sicherheit

Die Korrektheit und Integrität der Antworten kann hierbei noch weiter gesteigert werden, indem statt dem Namen einer Notary die Signatur über die Notary in die Berechnung des Hashs einbezogen wird. Hierdurch wird die Integrität der Implementierung der Notary-Klasse bzw. je nach Anbindung auch der Notary selbst in die Antwort mit einbezogen und kann leicht überprüft werden.