Hierarchische Authentifizierung mit SSL/TLS

Wolfgang Hüttenhofer

21.07.2010

Seminararbeit für
Konzepte von Betriebssystem-Komponenten

Informatik Lehrstuhl 4
Friedrich-Alexander Universität Erlangen-Nürnberg

Inhaltsverzeichnis

1 Einleitung

Zur sicheren Kommunikation über ein potentiell unsicheres Medium, wie z.B. das Internet, benötigt man sowohl eine Möglichkeit die Informationen zu verschlüsseln sowie ein System zur entfernten Authentifizierung. Dies kann zwar theoretisch von zwei vollkommen seperaten Systemen gehandhabt werden, jedoch werden diese beide meist zusammen erwünscht, sodass es sinnvoll ist eine generische Methodik zu entwickeln, die je nach Wunsch der Benutzer eine oder beide diese Aufgaben übernehmen kann.

Ein Ansatz zur Lösung dieser Problemstellung bietet das sogenannte Web of Trust, welches von Grund auf von den Benutzern selbst erstellt wird. Dies hat zwar Vorteile, wie einen sehr niedrigen Aufwand zur Aufstellung zu Beginn, sowie ein Ausbreiten des Verwaltungsaufwandes über alle Beteiligte, doch es bringt auch Nachteile mit sich. Diese sind unter anderem eine relativ lange Dauer vom ersten Einsatz bis eine weit verbreitete Nutzung möglich ist, sowie die Problematik, dass effektiv jeder Benutzer auch neue Mitglieder in das System mit aufnehmen können, was natürlich ein gewisses Sicherheitsrisiko darstellt.

Ein alternativer Ansatz ist deswegen ein hierarchischer Aufbau der Vertrauenshierarchie, auch PKI (Public Key Infrastructure) genannt, wobei nur gewisse Stellen neue Mitglieder akzeptieren können und dann für sie bürgen. Eine Implementierung diese Methode findet sich in dem weit verbreitetem kryptographischen Protokoll SSL (Secure Socket Layer) bzw. dessen Nachfolger TLS (Transport Layer Security), welches mit Hilfe von SSL-Zertifikaten zur Authentifizierung und einem Protoll zur Erstellung einer sicheren Verbindung über ein unsicheres Medium beide Probleme der Kommunikation addressiert.

Im folgenden wird zuerst das generelle Konzept der PKI und deren Infrastruktur erläutert, dann sowohl der Aufbau sowie der Verwendung von SSL-Zertifikaten betrachtet, und schliesslich im Detail der Aufbau einer sicheren SSL-Verbindung ohne Verwendung eines zweiten Kommunikationskanals erklärt.

2 Hauptteil

SSL bietet die Möglichkeit, eine sichere, verschlüsselte End-to-End Verbindung mit Authentifizierung einer oder beider Seiten aufzubauen, ohne feste Vorgaben an irgendeine Seite zu machen, außer dass sie das Protokoll implementieren. Es sitzt konzeptionell zwischen dem Application Layer und dem Transport Layer, was bedeutet, dass beliebige Application Layer Pakete verschlüsselt werden können. Es ist theoretisch mit jedem Transport Layer Protokoll verwendbar, macht aber einige Vorgaben wie dass z.B. kein Paketverlust auftritt sowie dass Pakete in der selben Reihenfolge angeliefert werden wie sie abgeschickt wurden - Garantien, die das TCP-Protokoll macht. In der praktischen Anwendung wird SSL deshalb so gut wie exklusiv mit TCP verwendet.

SSL unterstützt ebenso eine Vielzahl an Verschlüsselungs- und Kompressionsmethoden. Spezifisch ermöglicht es die Verwendung jeglicher beliebiger sogenannter *Cyphersuites*[1], vorausgesetzt sowohl Client als auch Server unterstützen diese. So kann z. B. in einer kommerziellen Anwendung, die SSL verwendet, eine private Implementierung von Verschlüsselung und Kompression eingesetzt werden, solange die entsprechenden Methoden mit der Software ausgeliefert werden.

1 Eine Cyphersuite ist eine Kombination aus Verschlüsselungsalgorithmus, Key-Exchange-Algorithmus und Message Digest Algorithmus

Das SSL-Protokoll sieht die Verwendung von symmetrischer und asymmetrischer Verschlüsselung vor. Da asymmetrische Verschlüsselung weit rechenaufwendiger als symmetrische Verschlüssung ist, wird erstere nur zur Erstellung eines gemeinsamen Geheimnisses eingesetzt, womit dann im Weiteren die eigentlichen Nutzdaten der Sitzung symmetrisch verschlüsselt werden.

SSL selbst wurde 1999 inoffiziell von TLS ersetzt, doch es bleibt in weitem Einsatz. TLS ist der „inoffizielle Nachfolger" von SSL, und wurde von der IETF (Internet Engineering Task Force) aufbauend auf SSL-Version 3 entwickelt. Die beiden Protokolle sind weitestgehend identisch, jedoch sind die Unterschiede groß genug, dass zwischen SSL und TLS keine Kompatibilität besteht. In dieser Arbeit kann allerdings davon ausgegangen werden dass eine Erläuterung von SSL ebenso für TLS gültig ist.

2.1 Public Key Infrastructure

In der PKI-Hierarchie sind gewisse Stellen besonders wichtig: die sogenannten *Certificate Authorities*, welche für das Überprüfen und Verifizieren neuer Mitglieder, sowie das Ausstellen neuer Zertifikate für diese, zuständig sind. Unter den Zertifikaten von besonderer Bedeutung sind sogenannte *Root Certificates*, auch als *Self-Signed Certificates* bekannt. Diese bilden die Basis der gesamten Hierarchie und sind deswegen für ihre Funktion absolut kritisch. Im folgenden betrachten wir diese etwas genauer, um schließlich die Hierarchie im ganzen überblicken zu können.

2.1.1 Certificate Authorities

Certificate Authorities, abgekürzt *CA*s, sind Mitglieder der Hierarchie, die befugt sind, neue Zertifikate für Mitglieder auszustellen. Da dies der kritischste Punkt für die Sicherheit des gesamten System ist, besteht die Aufgabe einer CA hauptsächlich darin, nur überprüften Leuten bzw. Firmen Zertifikate zu gewähren. Dies wird meist gelöst, indem ein Antragssteller zuerst bei einer von einer CA vertrauten Registrierungsstelle - *Registration Authority*, kurz *RA* genannt - ein Zertifikat beantragt. Die RA überprüft nun die Identität des Antragsstellers - bei Individuuen anhand eines anerkannten Ausweises wie Personalausweis, Pass, oder Führerschein, bei Firmen via eines intensiveren Verifizierungsprozesses. Nachdem dies erfolgreich durchgeführt wurde, reicht die RA den Antrag an die eigentliche CA weiter, welche nun das Zertifikat erstellt. Danach wird das neue Zertifikat mit dem privatem Schlüssel der CA signiert, womit später validiert werden kann, dass diese CA das Zertifikat ausgestellt hat, und dass es seitdem nicht geändert wurde. Schliesslich wird das signierte Zertifikat dem Antragssteller ausgehändigt. Meist gehören CA und RA zur selben Firma, damit sichergestellt ist dass die Aussagen der RA auch vertrauenswürdig sind.

Optional kann die CA nun auch Informationen über ausgestellte Zertifikate an eine *Validation Authortity* (*VA*) weiterreichen, bei der später bei der Überprüfung des Zertifikats die Echtheit noch einmal sichergestellt werden kann.

2.1.2 Root Certificates

Alle Hierarchien haben eine Wurzel, und im Falle von Public Key Infrastructure sind dies die *Root Certificates*. Dies sind Zertifikate, welche nicht von irgeneiner anderen CA signiert sind, sondern nur von sich selbst. Üblicherweise ist dies vollkommen unsicher, doch da die Hierarchie irgendwo beginnen muss, sind sie unumgänglich. Diese Root Certificates werden generell mit der Software, in der sie zu Verwendung kommen, ausgeliefert. Dies können z.B. Browser sein, aber häufig auch Home Banking Software oder ähnliche Anwendungen, welche Verschlüsselung und

Authentifizierung benötigen.

Da Root Certificates der Ausgangspunkt für das gesamte Vertrauen des Systems sind, und ein bösartiges Root Certificate potentiell weitreichende Auswirkungen haben kann, müssen Softwareentwickler sehr vorsichtig sein, welche Root Certificates sie mit in ihre Auslieferung mit aufnehmen. Dies führt zu einer sehr hohen Einstiegsschwelle für neue Certificate Authorities, da diese zuerst die großen Firmen (im Web meist Browserentwickler wie Microsoft, Mozilla oder Google) davon überzeugen müssen, dass sie vertrauenswürdig sind und ihr Root Certificate mit aufgenommen werden sollte.

2.1.3 Hierarchie

Abbildung 1: Beispiel einer PKI-Hierarchie[1]

Abbildung 1 zeigt beispielhaft die Struktur einer internen PKI-Hierarchie der Firma „Contoso". An oberster Stelle ist die „Contoso Root CA", welche ein Root Certificate besitzt. Dieses Root Certificate muss in allen Anwendungen, welche auf der Contoso PKI beruhen, enthalten sein. Die Root CA hat zwei Zertifikate ausgestellt - beide für weitere CAs.

Jede CA kann theoretisch eine beliebige Anzahl an Unterzertifikaten ausstellen, die alle optional selbst wieder eine CA zertifizieren können. In diesem Beispiel ist dies wieder der Fall - Contoso Marketing CA hat zwei Zertifikate für weitere CAs ausgestellt, Contoso Manufacturing CA nur eines. Diese Subordinate CAs sind nur dafür verantwortlich, die Endzertifikate für Benutzer auszustellen.

Zur Verifizierung der Endzertifikate benötigt man nur die jeweils darüberliegenden Zertifikate - für ein von der Contoso Marketing Subordinate CA-2 ausgestelltes Endzertifikat wäre dies das Zertifikat der Marketing Subordinate CA-2 selbst, der Marketing CA, und wie bereits erwähnt das Root CA Zertifikat, jedoch nicht die Zertifikate von Marketing Subordinate CA-1, Manufacturing CA, oder Manufacturing Subordinate CA-1.

2.2 SSL-Zertifikate

SSL-Zertifikate folgen dem X.509 Standard der ITU-T (Telekommunikations-Sektor der International Telecommunication Union)[2]. Es gibt bisher drei Versionen dieses Standards, jedoch werden so gut wie nur Version 1 und 3 verwendet. Dieser Abschnitt behandelt den Aufbau, die

Signierung, und die Validierung der Echtheit solcher Zertifkate.

2.2.1 Aufbau

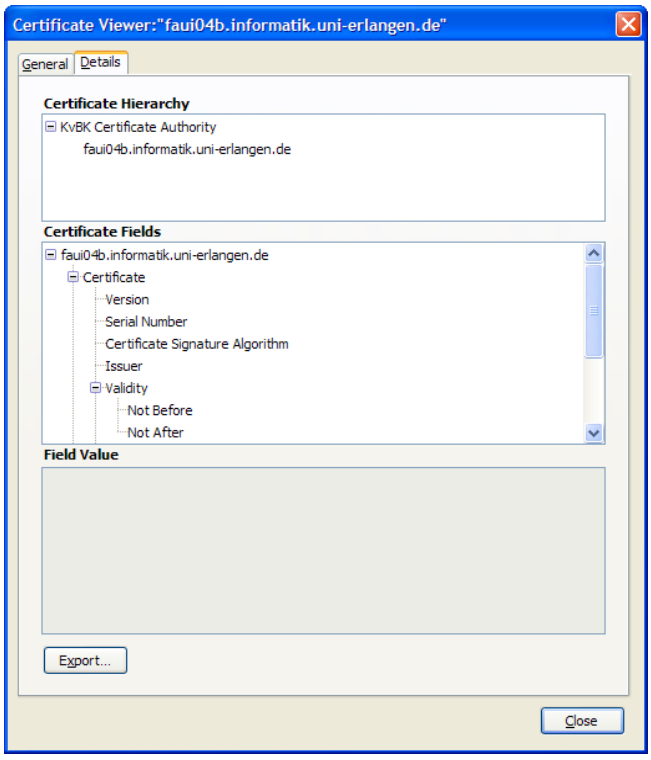

Abbildung 2: X.509 v1 Zertifikat (1)

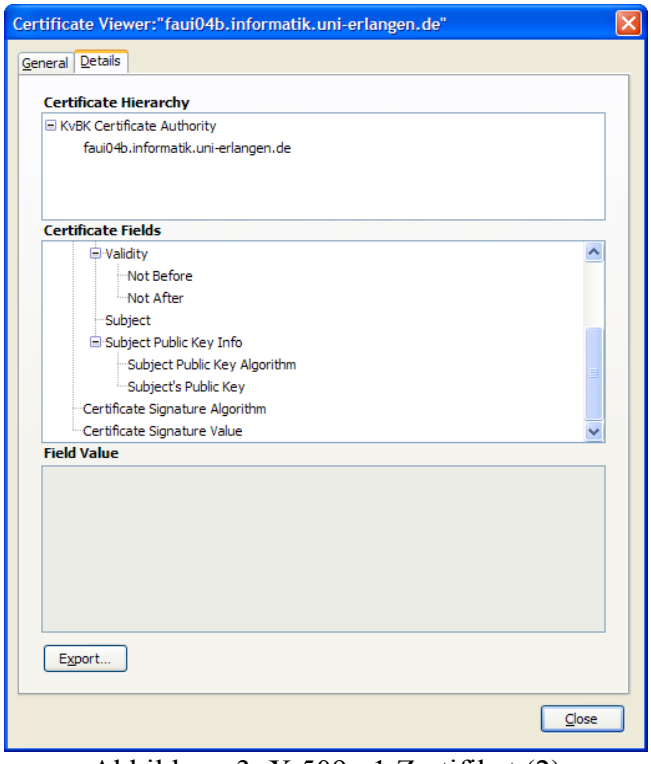

Abbildung 3: X.509 v1 Zertifikat (2)

Der Aufbau eines X.509-Zertifikats der Version 1 ist in Abbildung 2 & 3 beispielhaft dargestellt. In dieser Ansicht ist im oberen Feld die Zertifikatshierarchie des Zertifikats angegeben; in diesem Fall besteht sie nur aus dem Zertifikat selbst und der ausstellenden Root CA.

Felder eines X.509-Zertifikats sind:

- Version: 1-3, gibt verwendete Version des X.509-Standards an.
- Serial Number: Dem Zertifikat von der CA zugewiesene Seriennummer, sollte pro CA für jedes Zertifikat einzigartig sein.
- Certificate Signature Algorithm: Gibt an, mit welchem Algorithmus das Zertifikat signiert wurde. Sicherheitsfeld, um sicherzustellen dass ein Angriff nicht das Zertifikat verändert und mit einem unsichererem Algorithmus wieder verschlüsselt.
- Issuer: Ausstellende CA.
- Validity: Gültigkeitszeitraum des Zertifikats.
  - Not Before: Beginn des Gültigkeitszeitraumes
  - Not After: Ende des Gültigkeitszeitraumes
- Subject: Person oder Firma, welche von diesem Zertifikat authentifiziert wird.
  - CN/Common Name: Name der Person oder Firma
  - OU/Organizational Unit: Unterabteilung der Firma
  - O/Organization: Offizielle Bezeichnung
  - L/Location: Stadt oder genaue Adresse der Person oder Firma
  - S/State: Bundesstaat der Person oder Firma
  - C/Country: Land der Person oder Firma
- Subject Public Key Info
  - Public Key Algorith: Algorithmus, mit dem der Public Key zu verwenden ist.
  - Subject Public Key: Public Key der Person oder Firma
- Certificate Signature Algorithm: Wie oben.
- Certificate Signature: Signierungsschlüssel des Zertifikats

Version 3 des X.509-Standards erlaubt die Existenz zusätzlicher Felder, genannt *Extensions*. Diese geben z.B. an ob das Zertifikat als CA dienen kann, d.h. ob es selbst andere Zertifikate signieren darf oder nicht. Extensions können als *critical* markiert werden, was andeutet dass sie auf jeden Fall überprüft werden sollten. Dies ist jedoch der Endanwendung überlassen und wird nicht erzwungen. RFC 3280 definiert eine große Anzahl von Extensions für viele Anwendungsfälle, jedoch ist dies nicht restriktiv und Extensions können beliebig benannt und verwendet werden. Es ist in allen Fällen Aufgabe der Endanwendung, X.509 v3 Zertifikate auf Extensions zu überprüfen und dann aufgrund eigener Evaluation das Zertifikat anzunehmen bzw. abzulehnen.

2.2.2 Signieren von Zertifikaten

Um ein Zertifikat zu signieren, müssen folgende Vorraussetzungen erfüllt sein:

1. Die ausstellende CA muss bereits ein gültiges Zertifikat besitzen, evtl ein Root Certificate.
2. Die CA benötigt ihren privaten Schlüssel eines public/private Schlüsselpaares.
3. Es muss ein CSR (*Certificate Signing Request*) des Antragsstellers vorliegen - dies ist ein teilweise ausgefülltes, nicht signiertes Zertifikat.
4. Die Identität des Antragsstellers muss von der CA oder einer von ihr vertrauten RA bestätigt worden sein.

Der eigentliche Signiervorgang ist nun recht einfach:

1. Es muss ein Hash- und ein Verschlüsselungsalgorithmus gewählt werden.
2. In die beiden *Certificate Signature Algorithm* Felder des Zertifikats muss der gewählte Algorithmus eingetragen werden.
3. Es wird ein Hash über das Zertifikat, bis auf die beiden letzten Felder - *Certificate Signature Algorithm* und *Certificate Signature* - gebildet.
4. Dieser Hash wird mit dem Private Key der CA signiert und in das *Certificate Signature* Feld eingetragen.

Hiermit ist der Signiervorgang abgeschlossen und das Zertifikat zur Verwendung bereit.

2.2.3 Validieren von Zertifikaten

Um ein Zertifikat zu validieren, müssen folgende Vorraussetzungen erfüllt sein:

- Es muss das zu validierende Zertifikat vorliegen.
- Es müssen alle Zertifikate zwischen diesem und dem jeweiligen Root Certificate vorliegen.
- Das entsprechende Root Certificate muss bereits installiert sein.

Der eigentliche Validierungsvorgang ist nun recht einfach:

1. Die Signatur des zu validierenden Zertifikats wird mit Hilfe des im darüberliegenden Zertifkats angegebenen Algorithmus und Public Key entschlüsselt.
2. Es wird ein Hash über den Rest des Zertifikats, wieder mit Ausnahme der letzen beiden Felder, gebildet.
3. Dieser Hash wird mit dem in der Signatur angegebenen verglichen. Sind sie gleich, wurde das Zertifikat nicht bösartig modifiziert.
4. Nun müssen nur noch Subjekt, Gültigkeitsdauer und eventuell Extensions des Zertifikats überprüft werden.

Sollten keine Fehler aufgetreten sein, ist das Zertifikat gültig und das Subjekt authentifiziert.

2.3 SSL/TLS

Um die sichere Verwendung eines nicht vorher bestehenden gemeinsamen Geheimnisses für symmetrische Verschlüsselung zu ermöglichen, benötigt es einen Vorgang, der mit Hilfe von asymmetrischer Verschlüsselung ein solches Geheimnis auf sichere Weise erstellt und austauscht. Dies geschieht mit dem *SSL Handshake*.

Da ein solcher Handshake jedoch viel rechenaufwendige asymmetrische Verschlüsselung verwendet, und besonders bei Verwendung mit HTTP viele einzelne Verbindungen aufgebaut werden, ist es vorteilhaft, nur einmal ein Geheimnis zu erstellen und dieses fortan auch für neue Verbindungen bis zum Ende der Sitzung zu verwenden. Hierfür definiert das SSL-Protokoll einen sogenannten *Resume Handshake*.

2.3.1 SSL Handshake

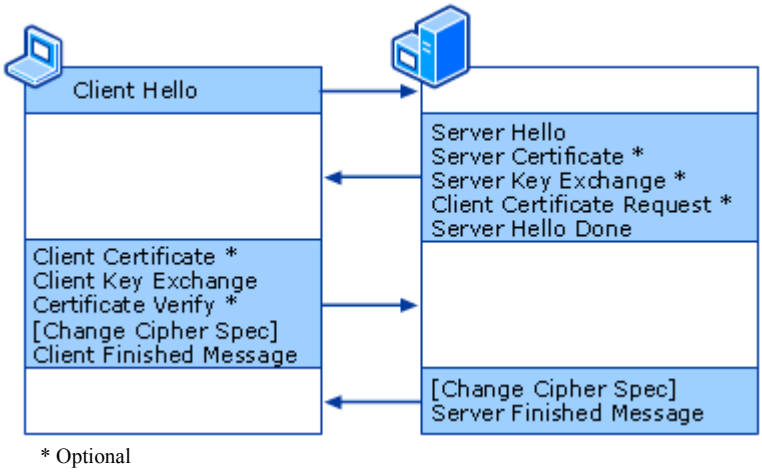

Abbildung 4: Der SSL Handshake[3]

Ein vollständiger SSL Handshake läuft wie folgt ab:

- Client Hello
  - Version Number: Die höchste SSL Version, die der Client unterstützt. TLS wird mit 3.1 bezeichnet.
  - Client Random: 4-byte Zahl bestehend aus Datum und Zeit, sowie einer 28-Byte Zufallszahl. Wird später zur Erzeugung des Master Secrets verwendet.
  - Cipher Suite: Eine Liste der Cyphersuites, die der Client unterstüzt. String bestehend aus jeweils einem asymmetrischen Verschlüsselungsalgorithmus, einem symmetrischen Verschlüsselungsalgorithmus, und einem Hash-Algorithmus.
  - Compression Algorithm: Liste von Kompressionsalgorithmen, die der Client unterstützt.

- Server Hello
  - Version Number: Der Server antwortet mit der höchsten SSL Version, die er selbst unterstützt. Diese Version wird verwendet.
  - Server Random: 4-byte Zahl bestehend aus Datum und Zeit, sowie einer 28-Byte Zufallszahl. Wird später zur Erzeugung des *Master Secrets* verwendet.
  - Cipher Suite: Der Server wählt aus der vom Client übergebenen Liste eine Cyphersuite aus und sendet diese zurück. Diese Cyphersuite wird verwendet.
  - Compression Algorithm: Der Server wählt aus der vom Client übergebenen Liste einen Kompressionsalgorithmus aus und sendet diesen zurück. Dieser Kompressionsalgorithmus wird verwendet.

- Server Certificate: Wird verwendet, falls der Server sich zum Client authentifizieren soll.
  - Server Certificate: Das eigentliche Zertifikat des Servers.
  - Validating Certificates: Liste von Zertifikaten zwischen dem Server-Zertifikat und einem Root Certificate.

- Server Key Exchange: Wird verwendet, falls der Server sich nicht zum Client authentifizieren soll.
  - Server Public Key: Der Public Key eines Public/Private Schlüsselpaares, meist zum Zeitpunkt der Verbindungsaufbaus erstellt und nach Ende verworfen.

- (Optional) Client Certificate Request: Wird verwendet, falls der Client sich auch zum Server authentifizieren soll.
  - Certificate Type: Liste von Algorithmen, mit denen das Client-Zertifikat verschlüsselt sein darf.
  - Accepted CAs: Liste von CAs, die vom Server als valide angesehen werden.

- Server Hello Done: Zeigt dem Client an, dass der Server fertig ist, und nun auf Antwort wartet.

- (Optional) Client Certificate: Nur verwendet, falls vorher ein Client Certificate Request empfangen wurde.
  - Client Certificate: Das eigentliche Zertifikat des Clients
  - Validating Certificates: Liste von Zertifikaten zwischen dem Client-Zertifikat und einem Root Certificate.

- Client Key Exchange: Der Client erzeugt aus seiner eigenen Random Nummer und der des Servers ein *Premaster Secret* und verschlüsselt es mit dem Public Key des Servers.
  - Client Version: Wiederholung der verwendeten SSL Version, zur Verteidigung gegen *Rollback-Angriffe*, welche die Version auf das weniger sichere SSL v2 heruntersetzen.
  - Pre-master Secret: Die mit dem Public Key des Servers verschlüsselte Zufallszahl.

- (Optional) Certificate Verify: Nur verwendet, falls ein Client Certificate gesendet wurde. Der Client bildet ein Hash aller bisherigen Nachrichten und signiert diesen mit seinem Public Key.
  - Hash: Der oben genannte, mit dem Public Key des Clients verschlüsselte, Hash.

- Change Cypher Spec: Teilt dem Server mit, das alle Nachrichten von nun an mit dem gemeinsamen Geheimnis verschlüsselt sind.

An dieser Stelle müssen sowohl Client als auch Server lokal aus dem Premaster Secret das Master Secret und sonstige Schlüssel erzeugen. Hierzu werden das *Premaster Secret*, das String-Literal „master secret” und die *Client Random* und *Server Random* Variablen an eine pseudorandom Funktion übergeben, welche dann das Master Secret zurückgibt. Hieraus werden nun das Client- und Server- *Write MAC Secret* und der Client- und Server- *Write Key* abgeleitet.

- Finished Message: Teilt dem Server mit, dass der Client mit der Berechnung fertig ist, und nun zum Informationsaustausch bereit ist. Diese Nachricht ist bereits mit dem *Client Write Key* verschlüsselt.

  - MAC Hash: Ein Hash bestehend aus einem Hash aller bisherigen Nachrichten und dem *Client Write MAC Secret*.

- Change Cypher Spec: Teilt dem Client mit, das alle Nachrichten von nun an mit dem gemeinsamen Geheimnis verschlüsselt sind.

- Finished Message: Teilt dem Client mit, dass der Server mit der Berechnung fertig ist, und nun zum Informationsaustausch bereit ist. Diese Nachricht ist bereits mit dem *Server Write Key* verschlüsselt.
  - MAC Hash: Ein Hash bestehend aus einem Hash aller bisherigen Nachrichten und dem *Server Write MAC Secret*.

Nun ist die Verbindung verschlüsselt, und der Informationsaustausch kann beginnen.

2.3.2 Resume Handshake

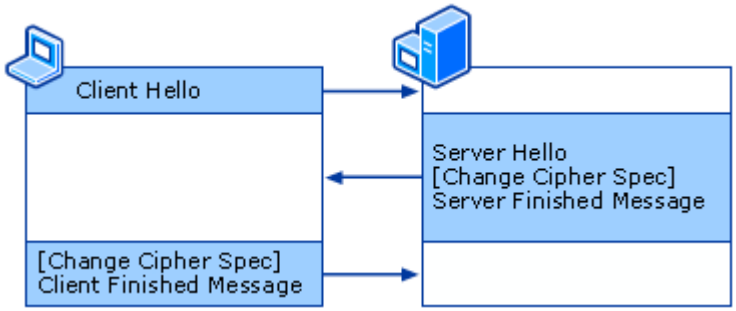

Abbildung 5: SSL Resume Handshake[3]

Der Resume Handshake ist eine abgekürzte Version des vollen Handshakes. Es werden hier nur die Unterschiede aufgelistet:

- Client Hello
  - (Optional) Session ID: Bei der ersten Verbindung einer Sitzung ist dieses Feld leer. Der Server sendet in seinem eigenen Hello eine ID mit, welche im Server Cache sowie lokal auf dem Client gespeichert wird. Falls der Client eine bestehende Sitzung wieder aufnehmen möchte, sendet er hier die gespeicherte Session ID mit.

- Server Hello
  - (Optional) Session ID: Findet der Server die ID nicht mehr in seinem Cache, so generiert er eine neue ID und schickt diese zurück. In diesem Fall geht der Client nun in einen vollen Handshake über. Falls der Server die ID noch bereit hat, schickt er diese in seinem Hello zurück, wodurch der Client weiß, dass der Server die benötigten Daten noch besitzt.

- Change Cypher Spec: Der Server geht sofort zum Verschlüsselungsbeginn über.

- Server Finished Message: Der Server benutzt den noch gespeicherten *Server MAC Write Key* und *Server Write Key* der vorherigen Verbindung mit dieser Session ID zum Verschlüsseln dieser Nachricht.
- Change Cypher Spec: Der Client geht ebenfalls sofort zum Verschlüsselungsbeginn über.

- Client Finished Message: Der Client benutzt den noch gespeicherten *Client MAC Write Key* und *Client Write Key* der vorherigen Verbindung mit dieser Session ID zum Verschlüsseln dieser Nachricht.

Nun ist die Verbindung verschlüsselt, und der Informationsaustausch kann beginnen. Zur Wiederaufnahme vorheriger Verbindungen ist dies weitaus praktischer, da nicht nur weniger Nachrichten ausgetauscht werden müssen, sondern auch jegliche Verwendung von asymmetrischer Verschlüsselung vermieden werden kann.

3 Zusammenfassung

Als zusammenfassender Rückblick soll hier noch einmal der Inhalt dieser Ausarbeitung kurz überschaut werden. Zuerst wurde die Public Key Infrastructure und ihr hierarchischer Aufbau konzeptuell erläutert und ihre wichtigsten Einrichtungen, wie *Certificate Authorities*, *Validation Authorities* und *Root Certificates* kurz umrissen. Danach wurde der Aufbau von SSL-Zertifikaten und der X.509-Standard dargestellt und die Signierungs- und Validierungsvorgänge beschrieben. Schliesslich wurde der SSL-Handshake erklärt und die Unterschiede des vollen Handshakes zum Resume Handshake aufgezeigt.

Während sowohl SSL-Zertifikate alleine gut zur Authentifizierung verwendbar wären, und der SSL-Handshake auch ohne jede Benutzung von Zertifikaten funktioniert, so ist doch die Verbindung dieser beiden Ansätze in ein einziges Protokoll weitaus simpler. Da SSL nicht an ein vorgegebenes Verschlüsselungs- oder Kompressions-schema gebunden ist, ist es weitreichend einsetzbar und erweiterbar.

Es gibt allerdings auch Probleme mit der Public Key Infrastructure. Allen voran steht momentan, dass es keinen standardisierten Weg gibt, Zertifikate zu widerrufen. Es gibt zwar hierfür zwei Ansätze - Certificate Revocation Lists (CRLs), von CAs publizierte Listen mit widerrufenen Zertifikaten, und das Online Certificate Status Protocol (OCSP), welches ähnlich zu Validation Authorities den Gültigkeitsstatus von Zertifikaten überprüft - jedoch wird keiner dieser beiden Ansätze im Moment eindeutig verfolgt. CRLs sind schwer zu finden und werden häufig zu selten erneuert, und bei dem OCSP wird die Antwort des Servers als authoritiv angesehen - was wiederrum zu Sicherheitsproblemen führen kann, falls diese Verbindung manipuliert werden kann.

Zur Zeit verlassen sich Entwickler und Anwender darauf, dass Certificate Authorities nur befugten Personen Zertifikate ausstellen. Dies geht nicht immer gut - siehe den Zwischenfall 2001 mit VeriSign und Microsoft - jedoch wurden bisher größere Auswirkungen vermieden. Allerdings ist dieses Vertrauen auf die Fehlerlosigkeit des Systems nach Ansicht des Autors das größte Problem, welches noch in der SSL-Implementierung der Public Key Infrastructure existiert, und wird in Zukunft sicher eines der wichtigsten Themen bei der Weiterentwicklung des Protokolls seien.

Literaturverzeichnis

[1] Webseite „Certificate Hierarchy (Windows)”, abgerufen 27.06.2010, http://msdn.microsoft.com/en-us/library/bb931353(VS.85).aspx

[2] PDF, „ITU-T Recommendation X.509: Public-key and attribute certificate frameworks”, abgerufen 27.06.2010, http://www.itu.int/rec/T-REC-X.509-200508-I/en

[3] Webseite „How TLS/SSL Works: Logon and Authentication”, abgerufen 27.06.2010, http://technet.microsoft.com/en-us/library/cc783349(WS.10).aspx