0.1 Theoretische Analyse des Protokoll-Overheads (Phase 2)

Meine hier vorgenommenen Untersuchung sollen erste Abschätzungen im Bezug auf die Netzwerklast erlauben und beschränken sich auf die Verkehrsanalyse zwischen zwei direkt miteinander verbundenen Knoten. Das Resultat dieses One-Hop-Szenario erlaubt es mir, die Netzwerklast für komplexere Topologien theoretisch zu bestimmen, indem man die Hop-Anzahlen der einzelnen Knoten zum Authentisierungsserver zusammenträgt und mit dem von mir ermittelten Wert (*oneHopOverhead*) multipliziert. Neben der Abschätzung der Gesammtnetzwerklast besteht ein weiterer Nutzen darin, die Netzwerklast auf den einzelnen Knoten abschätzen zu können, die im zentralisierten Netzwerk als Relay fungieren.

Sei $G = (V, E)$ ein ungerichteter Baum mit dem Authentisierungsserver als Wurzel, $r \in V$ ein dezidierter Relay-Knoten (über den die Kommunikation zum Authentisierungsserver läuft) bzw. der Authentisierungsserver selbst:

$$NodeOverhead(r) = oneHopOverhead \cdot \sum_{\substack{\exists q \in V: \\ (q,r) \in E, \\ q \neq r}} |(q, r)|$$

Nun zur Ermittlung des One-Hop-Overheads, also der Protokoll-Netzwerklast zwischen zwei Nachbarknoten. Die Phase 2 der MobiSEC-Architektur verwendet zwei Protokolle: Zum einen das TLS-Protokoll, um eine sichere Verbindung aufzubauen und zum zweiten das Schlüsselverteilungsprotokoll für das Mesh-Router-Netzwerk, auf die ich beide im Folgenden kurz eingehen möchte.

TLS bzw. das TLS Record Protokoll verwendet für den Betrieb drei Sub-Protokolle – Change-Cipher-Spec-Protokoll, Alert-Protokoll, Handshake-Protokoll – von denen allein das Handshake-Protokoll eine signifkante Netzwerklast (≥ 100[1] Byte) generiert und hier untersucht wird. In meiner Analyse habe ich für TLS zwischen *Session Setup Overhead* und *Session Resume Overhead* unterschieden. Der letzte Fall stellt eine Optimierung des Session Setups dar. Dabei arbeiten die Kommunikationspartner mit Session-IDs und sparen damit den Austausch von Zertifikaten ein. Um die TLS-Sicherheit an dieser Stelle durch die Langlebigkeit der Session nicht zu gefährden, müssen angemessene Zeithorizonte oder Ereignisbedingungen gewählt werden, um

[1] warum 100? //todo

ein neues Session Setup durchzuführen.

Aufbauend auf dieser TLS-Unterscheidung habe ich die Netzwerklast des Schlüsselverteilungsprotokolls untersucht und dabei zwischen *server driven* und *client driven* unterschieden.

Zur Berechnung der Netzwerklast des Schlüsselverteilungsprotokolls habe ich die folgenden Formeln verwendet:

$$\begin{aligned} kdpOverhead(serverDriven) &= (macLength + modeLength + keysInList * \\ &\quad keyLength + tstampLength) \\ kdpOverhead(clientDriven) &= (macLength + modeLength + seedLength + \\ &\quad tstampLength) \end{aligned}$$

Die verwendeten Richtwerte für TLS stammen sowohl aus der Spezifikation (Dierks und Allen, 1999) für TLSv1.0 als auch den Analysen von (Apostolopoulos et al., 1999). Für das Schlüsselverteilungsprotokoll aus (Martignon et al., 2009) habe ich gängige WEP-Schlüssellängen und UNIX-Timestamps verwendet. Das Ergebnis des One-Hop-Szenarios sei nachfolgend zusammenfassen:

Ausgangsparameter:

Keys in list:	15
Timeout for key usage:	60 sec
Session time:	900 sec

Netzwerklast bei einem normalen TLS-Session Setup:

Server Driven:	4544 Byte
Client Driven:	4312 Byte
Datenrate:[2]	147 Bytes/sec

Netzwerklast bei einem TLS-Session Resume:

Server Driven:	1544 Byte
Client Driven:	1312 Byte
Datenrate:	47 Bytes/sec

Estimated round trips: todo

[2] Durchschnittliche Datenrate relative zum Timeout.

Offene Fragen zur Bewertung: //todo

- Wie gut im Vergleich zu was? ACKS? Routing-Proto-Pkt? Hier Björn, Robert fragen..
- In welchen Zeitraum? Wie oft kann zwischen zwei TLS-Session Setups ein TLS-Session Resume stattfinden?
- Was bedeuten diese Zahl übertragen auf größere Netzwerke? Hier vielleicht eine kleine Modellrechnung mit Gittermodell als Netzwerktopologie

Literatur

Apostolopoulos, George; Vinod Peris und Debanjan Saha (1999): Transport Layer Security: How much does it really cost?

Dierks, T. und C. Allen (1999): The TLS Protocol. Version 1.0. RFC 2246.

Martignon, Fabio; Stefano Paris und Antonio Capone (2009): Design and implementation of MobiSEC: A complete security architecture for wireless mesh networks. *Elsevier*.