

HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

Mapping von KNX-Telegrammen für Nutzung verbreiteter Intrusion-Detection-Systeme wie Snort3

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von: Tom Schütt

geboren am:



geboren in:



Gutachter: Prof. Dr. Jens-Peter Redlich
Dr.-Ing. Siegmur Sommer

eingereicht am:

verteidigt am:

Inhaltsverzeichnis

1	Einführung	1
2	Motivation	2
3	Vorrangegangene Arbeiten	3
3.1	Diplomarbeit Alexander Fehr	3
3.2	Bachelorarbeit Vincent Weigt	4
4	Zielsetzung	4
5	Bestandteile der Bachelorarbeit	6
5.1	Implementation des vollständigen Mappings	6
5.2	Analyse und Zusatzfunktionen	7
6	Vergleich von IP mit KNX	7
6.1	Aufbau des IP-Headers	7
6.2	Aufbau von KNX Telegrammen	10
6.2.1	KNX-Acknowledgements	10
6.2.2	L_Poll_Data	11
6.2.3	L_Data_Frame	11
6.2.4	L_Data_Standard	12
6.2.5	L_Data_Extended	14
6.3	cEMI Frame	14
6.4	Implikationen für das Mapping	15
7	Übersicht	16
7.1	Physische Komponenten	17
7.2	Software Komponenten	17
8	Aufsetzen der Komponenten	18
8.1	KNX-Installation	18
8.2	Reiner SCT cyberJack Kartenleser BSI-K-TR-0067-2010	19
8.3	Mappingsoftware	19
8.4	Java API	19
8.5	Snort3	19
9	Aufbau der Mapping Software	20
10	Erklärung der Module	21
10.1	config.ini	21
10.2	keystore_manager	22
10.3	KNX_Telegramme	22
10.3.1	KNX_TP1_Telegramm	23

10.3.2	KNX_IP_cEMI_Frame	24
10.4	util_general	24
10.5	util_config	25
10.6	errors	25
10.7	TunTap	26
10.8	KNXasIPFactory	27
10.8.1	map_incoming_traffic_from_eth	27
10.8.2	map_incoming_traffic_from_USB	27
10.9	main	27
10.10	testing	28
11	Tests	28
11.1	Funktionstest über die USB Bridge	28
11.2	Funktionstest über den KNX/IP Router, inklusive Ansprache des sicheren Schlüsselspeichers	29
11.3	Lasttest unter Verwendung des testing Moduls	29
12	Ergebnis	31
13	Ausblick	33
13.1	Erweiterung des Mappers	33
13.2	Weiterführende Tests	34
14	Glossar	35

1 Einführung

KNX ist ein offener Standard für Gebäudeautomation. Er findet vorrangig in der Industrie, aber auch in privaten Haushalten Anwendung, wenn es darum geht, mehrere räumlich nah beieinander liegende Sensoren und Aktoren zur Hausgerätesteuerung miteinander zu verbinden. Die Geräte eines Netzes werden als eine (KNX-)Installation bezeichnet. Der Standard spezifiziert dabei sowohl, über welche Medien verschiedene Gerätschaften einer Installation miteinander kommunizieren können, als auch den genauen Aufbau der Nachrichten. Das am meisten verbreitete Kommunikationsmedium für KNX ist die Verkabelung über den Twisted-Pair-Bus (KNX-TP bzw. KNX-TP1), es wird aber auch die Kommunikation über Radio Frequency (RF) oder Powerline unterstützt [Auta].

Bei ihrer initialen Veröffentlichung waren KNX Netzwerke dabei noch recht isoliert. Zwar waren die Geräte alle untereinander verbunden, jedoch geschah dies zumeist innerhalb eines Gebäudes über einen TP-Bus, sodass von außen kein Zugriff auf das Netz bestand und der Netzwerkverkehr im Allgemeinen erst einmal nicht mitgehört werden konnte. Deshalb ist in der Spezifikation der Kommunikation über TP keine Verschlüsselung vorgesehen [Ass21b].

Mit der zunehmenden Digitalisierung und der Verbreitung des Internetprotokolls (IP) entschied sich die KNX Association ihren Standard um KNX IP zu erweitern, womit Schnittstellen zum Internet definiert wurden. Darüber wurde es auch ermöglicht, mehrere regional verteilte KNX-Installationen zusammenzuschließen [Ass21c].

Gleichzeitig wurden dadurch aber die ehemals eher isolierten Netzwerke offener und somit auch anfälliger für Angriffe von außen. So berichtet etwa die Limes Security über einen Angriff, bei dem der Zugang zu einem Büro durch Manipulieren der vor Ort installierten KNX-Komponenten blockiert wurde. Der Angreifer benötigte hierfür keinen direkten Zugriff auf die Hardware [Sec].

Doch selbst bei Installationen, die nicht an das Internet angebunden sind, existieren effektive Angriffe. Ein Beispiel dafür wäre die Software Erebos von der Antago GmbH, welche das Fernsteuern von Geräten ermöglicht, die an den KNX-Bus angeschlossen sind. Hierfür benötigt der Angreifer Zugang zum Bus [Dör] [Gmb]. Um solchen Angriffen vorzubeugen, erweiterte die KNX Association ihren Standard erneut, diesmal um KNX Data Secure und KNX IP Secure [Assa] [Assb]. In diesen Erweiterungen führten sie die Verschlüsselung der Anwendungsdaten und eine Authentifikation der gesendeten Telegramme durch MACs ein.

Unabhängig von der Entwicklung von KNX kam es bereits in den achtziger Jahren zur Einführung der ersten Intrusion Detection Systeme (im Folgenden IDS). Dabei handelt es sich um Programme, welche den Datenverkehr eines Netzwerkes überwachen. Je nach Ausrichtung kann diese Überwachung nur gegen ein- bzw. ausgehende Nachrichten sein (Host-based intrusion detection system, kurz HIDS) oder auch den Verkehr innerhalb eines Netzwerkes überwachen (Network Intrusion Detection System, kurz NIDS).

Die überwachten Nachrichten werden dann vom IDS überprüft. Dabei kommen hauptsächlich zwei Verfahren zum Einsatz, die signaturbasierte und die anomaliebasierte Gefahrenerkennung. Bei der signaturbasierten Gefahrenerkennung werden die Nachrichten mit einem festen Satz von Regeln verglichen. Dadurch können bereits be-

kannte Angriffe verlässlich erkannt werden. Der andere Ansatz ist der anomaliebasierte, welcher maschinelles Lernen nutzt, um ungewöhnliches Verhalten zu erkennen. Wird dabei eine Bedrohung erkannt, können die auffälligen Nachrichten je nach Konfiguration dabei einfach protokolliert, vollständig blockiert, oder ein Administrator alarmiert werden.

Um diese Funktionalität umzusetzen, muss ein IDS Zugriff auf alle Nachrichten im Netzwerk haben. Häufig ist es notwendig, die Nachricht im Klartext vorliegen zu haben. Hierfür muss gegebenenfalls eine Entschlüsselung der Nachricht vorgenommen werden. Um sicherzustellen, dass die Nachricht authentisch ist, muss zudem die Prüfung des Message Authentication Codes (MAC) erfolgen.

2 Motivation

Mittlerweile gibt es sowohl für Hardware wie auch Software, kostengünstige und leicht erhältliche Methoden, wie die KNX-USB-Bridge oder den KNX/IP-Router, womit es möglich wird eigene Telegramme an eine Installation zu senden. Dies erhöhte, zusammen mit der erhöhten Erreichbarkeit der Installationen über das Internet, die Anfälligkeit gegen Angriffe. Dem wurde zwar mit der Einführung von KNX Data Secure entgegengewirkt, jedoch ist es wünschenswert, diesen Schutz um die Funktionen eines IDS zu erweitern.

Hierfür gibt es mehrere Möglichkeiten. Zum einen können bei den IP-Netzwerken, die mit einem KNX-Netzwerk verbunden sind, herkömmliche IDS verwendet werden. Dadurch würde zwar die KNX-Installation selbst nicht geschützt, sie könnte aber indirekt von der zusätzlichen Sicherheit in den angrenzenden Netzwerken profitieren. Allerdings setzt dies auch voraus, dass das angebundene Netzwerk selbst vertrauenswürdig ist und dass es keine anderen Anbindungen gibt, welche das IDS umgehen. Angriffe durch direkten Buszugriff könnten ebenfalls nicht erkannt werden. Zudem müsste die Verwendung von KNX IP Secure gesondert betrachtet werden, da dieses verschlüsselte KNX-Telegramme als Payload überträgt, welche somit nicht untersucht werden könnten. Das IDS benötigt für diesen Fall also eine Möglichkeit, die KNX-Telegramme zu entschlüsseln.

Ein anderer Ansatz wäre es, ein spezielles IDS für KNX zu entwickeln, welches als NIDS alle Nachrichten im Netzwerk mitliest. Dies verspräche eine tiefe Integration und vollständige Abdeckung der KNX-Installation, wobei explizit auf Eigenheiten des Standards eingegangen werden kann und externen Systemen kein Vertrauen entgegengebracht werden muss. Als Folge dessen wäre ein hoher Schutzfaktor zu erwarten. Allerdings wäre dieser Ansatz auch mit großem Entwicklungsaufwand verbunden, da das entsprechende IDS von Grund auf entwickelt werden müsste.

Daher soll in dieser Arbeit ein dritter Ansatz untersucht werden. KNX und IP sind beides Protokolle zur Übertragung von Nachrichten und weisen deshalb einige Gemeinsamkeiten auf. Dazu zählen eindeutige Geräteadressen, Time-to-live-Mechanismen und Prüfsummen. Auch wenn sich die Implementation zwischen den Standards unterscheidet, könnte es möglich sein, die Parameter des einen Standards auf die des anderen abzubilden und dabei eine vergleichbare Funktionalität zu erhalten. Es könnte dadurch ebenfalls möglich werden, dass ein IDS, welches eigentlich für die Analyse von IP-Paketen entwickelt wurde, auch in der Lage ist, Bedrohungen in abgebildeten KNX-Telegrammen zu erkennen. Dies

würde erlauben, das KNX Netzwerk direkt zu überwachen, ohne dabei ein gänzlich neues IDS entwickeln zu müssen. Im Fall einer erfolgreichen Abbildung könnte somit der Schutz von KNX-Installationen durch verhältnismäßig geringen Entwicklungsaufwand signifikant erhöht werden.

3 Vorrangegangene Arbeiten

Der Ansatz, IP basierte IDS zur Analyse von KNX-Installationen zu nutzen, ist bereits in zwei anderen Arbeiten untersucht worden.

Zunächst die Diplomarbeit von Alexander Fehr [Feh20]. In ihr wird analysiert, wie KNX Data Secure seine Sicherheitsziele implementiert. Darauf basierend entwirft sie die Grundlagen für ein IDS, dessen zentrale Komponente ein zentraler, sicherer Schlüsselspeicher ist.

Die zweite Arbeit ist die Bachelorarbeit von Vincent Weigt [Wei24]. Diese stellt eine praktische Implementation des von Fehr konzipierten sicheren Schlüsselspeichers vor und beschreibt dessen Funktionen und Schnittstellen ausführlich.

Da sich diese Arbeit an die beiden vorangegangenen anschließt, werden die relevanten Bestandteile im Folgenden zusammengefasst.

3.1 Diplomarbeit Alexander Fehr

KNX Data Secure garantiert Datenauthentizität und -integrität, Vertraulichkeit und Frische [Feh20, Abschnitt 2.2]. Während der Einrichtung der Installation wird zu jedem Kommunikationskanal ein sogenannter Laufzeitschlüssel erstellt, welcher auf den Geräten gespeichert wird, die den Kanal nutzen. Dieser wird sowohl für die Berechnung von MACs als auch die Verschlüsselung der Telegramme genutzt [Feh20, Abschnitt 2.2]. Die Einhaltung dieser Sicherheitsziele hängt somit an der Geheimhaltung der symmetrischen Laufzeitschlüssel.

Für die Kommunikation zwischen zwei Geräten werden dabei die beiden Laufzeitschlüssel der miteinander kommunizierenden Geräte genutzt. Dadurch sind die Schlüssel über alle Geräte verteilt. Falls ein Angreifer Zugriff auf ein KNX-Gerät und dessen Schlüssel erhalten sollte, ist somit nur ein Teil der gesamten Installation kompromittiert.

Damit die Nachrichten jedoch von einem IDS untersucht werden können, ist es nötig, sie vorher zu entschlüsseln und gegebenenfalls die MACs zu prüfen. Das IDS muss also in der Lage sein, auf alle Laufzeitschlüssel zuzugreifen [Feh20, Abschnitt 5.1.1]. Um dies zu ermöglichen, werden die Schlüssel nach der Installation in einen zentralen Schlüsselspeicher exportiert.

Falls ein Angreifer Zugriff auf den Schlüsselspeicher und somit auch auf die Laufzeitschlüssel erhielte, wären allerdings sämtliche Schutzziele kompromittiert. Daher muss Schlüsselspeicher speziell abgesichert sein. In der Diplomarbeit werden hierzu mehrere Varianten untersucht und verschiedene kryptografische Hardwarekomponenten verglichen. Die konkrete Implementation erfolgt durch die Bachelorarbeit von Vincent Weigt, weshalb

sie hier nicht weiter ausgeführt werden soll. Als zentrale Recheneinheit für das IDS wird ein Raspberry Pi vorgeschlagen.

3.2 Bachelorarbeit Vincent Weigt

Die Arbeit besteht aus der Software für den sicheren Schlüsselspeicher [Wei], sowie deren Dokumentation und Analyse [Wei24].

Der Schlüsselspeicher ermöglicht die Entschlüsselung und Verifikation von Telegrammen über eine API. Dabei kommt eine Javacard zum Einsatz, welche über einen Kartenleser an das IDS angebunden werden kann [Wei24, Abschnitt 1].

Um den Schlüsselspeicher zu verwenden, muss er zunächst initialisiert werden. Es wird davon ausgegangen, dass dieser Vorgang in einer sicheren Umgebung stattfindet. Bei der Initialisierung wird ein Wrapping-Schlüssel auf der Javacard erzeugt. Dieser verbleibt immer auf der Javacard und wird nie über die API ausgegeben. Er wird anschließend genutzt, um die Laufzeitschlüssel der KNX-Anwendung zu verschlüsseln. Die so verschlüsselten Laufzeitschlüssel können anschließend im Dateisystem des Raspberry Pi abgelegt werden. Während der Laufzeit ist es dann nicht mehr möglich auf den Wrappingschlüssel zuzugreifen [Wei24, vgl. Abschnitt 1.1]. Wenn nun ein verschlüsseltes Telegramm vom IDS analysiert werden soll, kann nun die API des sicheren Schlüsselspeichers angesprochen werden. Diese übergibt das Telegramm zusammen mit seinem verschlüsselten Laufzeitschlüssel an die Javacard [Wei24, Abschnitt 2.3.3 `INS_DECRYPT_AND_VERIFY`].

Auf dieser wird dann der Laufzeitschlüssel entschlüsselt und anschließend genutzt, um die Verschlüsselung des Telegramms aufzuheben, oder dessen MAC zu verifizieren. Das entschlüsselte Telegramm, nicht aber der Laufzeit- oder Wrappingschlüssel, werden anschließend zurück an die API und dann an das IDS weitergeleitet. Solange die Schutzfunktionen der Javacard nicht gebrochen werden können, stellt dieser Ablauf somit sicher, dass ein Angreifer nie an die Laufzeitschlüssel gelangen kann. Selbst bei physischem Zugriff auf die Javacard ist es nicht möglich, den Wrappingschlüssel zu extrahieren.

Für das IDS wird zur Laufzeit hauptsächlich die Operationen `INS_DECRYPT_AND_VERIFY` verwendet [Wei24, Abschnitt 2.3.3 `INS_DECRYPT_AND_VERIFY`]. Diese gibt, wenn das Telegramm erfolgreich entschlüsselt werden kann und die MAC korrekt ist, den entschlüsselten Payload des Telegramms zurück. Falls die MAC-Prüfung fehlschlägt, wirft die Operation eine entsprechende Exception. In diesem Fall wird stattdessen die Operation `insDecrypt` genutzt, welche den entschlüsselten Payload ohne Prüfung der MAC zurückgibt.

Der größte Nachteil der Arbeit liegt darin, dass die Entschlüsselung durch die Javacard verhältnismäßig zeitaufwendig ist. Bei der aktuellen Implementation kann also nur ein Teil des maximalen Durchsatzes eines KNX-Busses in Echtzeit analysiert werden. Dies lässt sich allerdings durch die Verwendung mehrerer Javacards beheben.

4 Zielsetzung

Das Ziel dieser Arbeit ist zu untersuchen, inwiefern es durch das Mapping (im Deutschen Abbilden) von KNX-Telegrammen auf IP-Pakete möglich ist, herkömmliche IDS für die

Überwachung von KNX-Netzen zu verwenden.

Hierfür soll ein Proof of Concept (im Folgenden PoC) geschaffen werden, welcher nachweist, dass der Ablauf des Mappings erfolgreich durchlaufen werden kann. Dieser besteht daraus, dass Telegramme von einem KNX Netzwerk an einen Computer übertragen werden. Auf dem Computer soll eine Mappingsoftware laufen, welche zu jedem erhaltenen Telegramm eine Abbildung in Form eines IP-Paketes erzeugt. Falls nötig, soll der Computer dabei einen sicheren Schlüsselspeicher nutzen, um an den Klartext der erhaltenen Telegramme zu gelangen. Die abgebildeten Telegramme sollen dann an ein IDS übergeben und von diesem geloggt werden.

Die Implementierung erfolgt dabei auf einem Raspberry Pi 4 Model B mit 8 GB RAM. Dies ist ein verhältnismäßig leistungsschwacher Computer, der sich allerdings aufgrund seiner geringen Größe und niedrigen Preises, gut für die Entwicklung von Prototypen eignet [Fou]. Außerdem kommt der Raspberry Pi wegen seines geräuschlosen Betriebs und seinem vergleichsweise niedrigen Stromverbrauch auch außerhalb des Prototyping als mögliche Hardware für eine industrielle Implementation des IDS infrage. Somit ist es nützlich, dieses Potenzial mitzutesten.

Snort ist ein IDS, welches sowohl auf Unix- als auch Windowssystemen funktioniert. Da es zusätzlich im Feld eine hohe Verbreitung besitzt, kostenfrei und darüber hinaus quelloffen ist, wird es in dieser Arbeit exemplarisch für alle IDS verwendet. Da es die neueste der veröffentlichten Versionen ist, wird Snort3 genutzt.

Bei der Implementierung des Mappings soll allerdings darauf geachtet werden, eine möglichst allgemeine Schnittstelle bereitzustellen, damit diese in Zukunft auch mit anderen IDS genutzt werden kann. In dieser Arbeit werden allerdings keine weiteren IDS verwendet.

An die Implementierung des PoC werden die folgenden Anforderungen gestellt:

1. Die Funktionsweise des KNX-Netzes darf nicht beeinträchtigt werden.
2. Die Sicherheit der Verschlüsselung darf durch die Mappingsoftware nicht gefährdet werden. Insbesondere muss sichergestellt sein, dass ein Angreifer keinen Zugriff auf die KNX-Schlüssel erhalten kann.
3. Die Abbildung der Pakete soll in Echtzeit erfolgen.
4. Die Abbildung soll sowohl für verschlüsselte, MAC-authentisierte Telegramme als auch für solche im Klartext funktionieren. Die abgebildeten Telegramme sollen dem IDS im Klartext vorgelegt werden. KNX liefert über den KNX/IP-Router bereits die Möglichkeit, Netzwerkverkehr im common EMI Format (cEMI) zu übertragen. Dabei werden alle relevanten Datenfelder als UDP-Payload verpackt. Somit besteht bereits eine Möglichkeit, Intrusion Detection zu verwenden. In diesem Fall ist es allerdings notwendig, ausführliche Regeln zu schreiben, welche einzelne Datenfelder aus dem Payload extrahieren und daraus ihre Schlüsse ziehen. Ziel der Implementation ist daher, eine möglichst große Menge von Informationen aus dem ursprünglichen Telegramm auf geeignete Felder im IP-Header zu übertragen und somit unmittelbar für das IDS zugänglich zu machen.

Zusätzlich werden Funktionen aus KNX in Bezug auf ihre Sicherheitsrelevanz und Umsetzbarkeit untersucht, welche sich nicht in IP abbilden lassen, sowie ein Ausblick gegeben, wie das Mapping noch vertieft werden könnte.

Das Ziel ist dabei nicht, ein vertriebsfähiges IDS für KNX-Netzwerke zu entwickeln. Die Implementation fungiert lediglich als PoC und soll die Umsetzbarkeit, sowie auftretende Probleme ermitteln.

5 Bestandteile der Bachelorarbeit

In diesem Abschnitt soll die Implementation des Mappings in einzelne Arbeitsschritte aufgeteilt und diese erläutert werden.

5.1 Implementation des vollständigen Mappings

Zunächst einmal müssen die bereits erstellten Hard- und Softwarekomponenten eingerichtet werden.

Als Erstes soll dabei eine KNX-Installation geschaffen werden, die den zum Testen notwendigen Datenverkehr generieren kann.

Anschließend muss der sichere Schlüsselspeicher auf dem Raspberry Pi und der Javacard installiert werden.

Zudem muss Snort3 installiert werden und die notwendigen Konfigurationen vorgenommen werden, damit ein Netzwerkinterface überwacht und der darüber erhaltene Datenverkehr geloggt werden kann.

Zuletzt muss eine Mappingsoftware geschrieben werden, welche die Telegramme der KNX-Installation aufgreift und in IP-Pakete umwandelt. Dabei soll darauf geachtet werden, dass die transformierten Pakete in ihrer Funktion möglichst nah an den ursprünglichen Telegrammen bleiben.

Da KNX und IP teilweise unterschiedliche Mechanismen für die Übertragung von Nachrichten vorsehen, werden voraussichtlich nicht alle Funktionalitäten exakt übertragbar sein. Dies bedeutet, dass die Standards KNX und IP zunächst gegeneinander abgeglichen werden müssen, um festzustellen, welche Parameter sich abbilden lassen.

Außerdem müssen Schnittstellen zu den anderen Komponenten geschaffen werden.

Dies bedeutet, dass Möglichkeiten geschaffen werden müssen, Telegramme des KNX-Netzwerks an den Raspberry Pi zu übertragen. Hierbei sollen sowohl KNX/IP Lösungen unterstützt werden, als auch das direkte Abgreifen der Telegramme vom Bus. Da auf dem Bus ein anderes Protokoll verwendet wird als für KNX/IP, muss die Mappingsoftware in der Lage sein, sowohl common EMI Telegramme (für KNX/IP) als auch L_Data_Standard und L_Data_Extended Formate (vom Bus) zu mappen.

Darüber hinaus muss die Mappingsoftware in der Lage sein, den sicheren Schlüsselspeicher anzusprechen. Hierfür ist die von Vincent Weigt bereitgestellte API geeignet [Wei].

Zu guter Letzt muss eine virtuelle Ethernet-Schnittstelle geschaffen werden, über die Snort die gemappten Pakete empfangen kann.

5.2 Analyse und Zusatzfunktionen

Sobald die Implementation der Mappingsoftware abgeschlossen ist und das erfolgreiche Zusammenspiel der verschiedenen Komponenten bestätigt werden konnte, soll theoretisch analysiert werden, ob Lücken in der Gefahrenerkennung des IDS auftreten, wie relevant diese gegebenenfalls sind und welche Schritte unternommen werden können, um ihnen entgegenzuwirken.

Da viel Aufwand für die Konfiguration der verschiedenen Bestandteile aufgewandt wurde, sollen die vorgenommenen Schritte dokumentiert werden, um die Reproduzierbarkeit zu gewährleisten und eine etwaige Weiterentwicklung zu erleichtern.

Die geschaffene Mappingsoftware wird detailliert beschrieben, wobei sowohl die einzelnen Funktionen und deren Zusammenspiel, als auch deren Relevanz für das Mapping bestimmter Parameter erklärt wird.

Es wurde außerdem ein Testmodus implementiert, der genutzt werden kann, um die Applikation während der Entwicklung zu testen. Insbesondere wird die Abhängigkeit zu einer physischen KNX-Installation dadurch ausgesetzt.

Der Testmodus kann außerdem genutzt werden, um manipulierte Telegramme mappen zu lassen, wodurch Angriffe auf das Netzwerk simuliert werden könnten.

6 Vergleich von IP mit KNX

Dieser Abschnitt beschäftigt sich mit den Gemeinsamkeiten und Unterschieden der beiden Netzwerkstandards. Der größte Unterschied besteht dabei darin, dass KNX für verschiedene Transportmedien und unterschiedliche Formate nutzt, während IP unabhängig von der physischen Schicht immer den gleichen Aufbau besitzt. Da in dieser Arbeit ausschließlich KNX/TP, sowie KNX/IP, für die Übertragung zum Raspberry Pi, verwendet werden, werden auch nur diese Protokolle verglichen. Allerdings nutzt KNX auch innerhalb eines Mediums teilweise verschiedene Formate, um jeweils verschiedene Anforderungen zu erfüllen.

Daher wird zunächst der Aufbau des IP-Headers und den darauf aufbauenden Netzwerkschichten beschrieben und dann nacheinander der Vergleich zu den verschiedenen Varianten von KNX gezogen.

6.1 Aufbau des IP-Headers

Der IP Header besitzt zunächst einige Felder, die sehr spezifische Werte erwarten und daher nur bedingt nutzbar für ein Mapping sind. Diese sind nach [Gor17, Seite 202-204]:

- *Versionsnummer*:
 - Beschreibung: Gibt die genutzte IP Version an.
 - Länge: 4 Bit

- *Headerlänge:*
 - Beschreibung: Die Header Länge gibt an, wie viele 4 Byte Blöcke im Header enthalten sind. Da Nachrichten mit falscher Längenangabe nicht korrekt interpretiert werden können, wird dieser Parameter korrekt gesetzt.
 - Länge: 4 Bit
- *Paketlänge:*
 - Beschreibung: Die Paketlänge gibt die Länge des gesamten Pakets an. Da eine Manipulation dieses Wertes zu einer falschen Auswertung durch den Empfänger führen würde, wird dieser Wert korrekt gesetzt.
 - Länge: 16 Bit
- *Identification:*
 - Beschreibung: Die Identification wird genutzt, um die Reihenfolge von Paketen bei Fragmentierung zu kodieren. Da KNX-Telegramme deutlich kleiner sind, als die zulässigen Paketlängen in IP oder UDP, wird diese Funktionalität beim Mapping jedoch nicht benötigt.
 - Länge: 16 Bit
- *Flags:*
 - Beschreibung: Die Flags markieren, ob das Paket fragmentiert werden darf oder nicht und ob noch weitere Fragmente folgen. Da wir davon ausgehen, dass die Pakete nicht fragmentiert werden, setzen wir die Flags so, dass das Paket nicht fragmentiert werden darf und keine weiteren Fragmente folgen.
 - Länge: 3 Bit
- *Fragment Offset:*
 - Beschreibung: Der Fragment Offset wird ebenfalls für Fragmentierungen benötigt.
 - Länge: 13 Bit
- *TTL:*
 - Beschreibung: Die Time to Live gibt an, wie oft ein Paket weitergeleitet werden darf. Bei jeder Weiterleitung muss die TTL um eins reduziert werden. Dies soll verhindern, dass Pakete bei Kreisschlüssen beliebig lange rotieren und somit die Bandbreite auslasten können.
 - Länge: 8 Bit

- *Protocol:*
 - Beschreibung: Das Protocol gibt an, welches Protokoll für Layer 4 verwendet wird. Da der größte Teil von KNX TP verbindungslos ist, wird in dieser Arbeit UDP anstelle von TCP verwendet. Im Ausblick wird diese Thematik weiterführend behandelt.
 - Länge: 8 Bit
- *Options:*
 - Beschreibung: Das Optionsfeld wird heutzutage eigentlich nicht mehr verwendet. Da sein Inhalt somit nur schwach definiert und für IDS schwer einzuordnen ist, wird dieses Feld nicht für die Mappingsoftware genutzt. Stattdessen werden die Daten, falls nötig, im Payload abgelegt.
 - Länge: Variabel

Die restlichen Felder des IP Headers sind weniger eingeschränkt und sollen daher genauer betrachtet werden:

- *Type of Service:*
 - Beschreibung: Der Type of Service enthält Parameter, wie das Paket geroutet werden soll. In der Praxis wird diese Funktion allerdings kaum verwendet. Daher wurde das Feld neu definiert und enthält heutzutage meistens den *Differentiate Services Code Point* [Mik], sowie die Explicit Congestion Notification (ECN) [Autb]. Es wird genutzt, um verschiedene Anwendungen nach ihrer Dringlichkeit priorisieren zu können und Überlast zu signalisieren. Die Nutzung dieser Felder ist allerdings Optional.
 - Länge: 8 Bit
- *Header Prüfsumme:*
 - Beschreibung: Die Header Prüfsumme, wird genutzt um die korrekte Übertragung des Pakets zu sichern. Sie wird zunächst vom Sender berechnet und in den Header eingefügt. Der Empfänger sowie alle Zwischenstationen berechnen nach Empfang ihrerseits die Prüfsumme und verwerfen das Paket, falls die Prüfsummen nicht übereinstimmen.
 - Länge: 16 Bit
- *Quelladresse:*
 - Beschreibung: Die Quelladresse gibt an, welches Gerät eine Nachricht versendet hat. Die IP Adresse eines Gerätes oder Netzwerks sollte im Idealfall eindeutig sein.
 - Länge: 32 Bit

- *Zieladresse:*
 - Beschreibung: Die Zieladresse gibt an, welches Gerät eine Nachricht empfangen soll. Die IP Adresse eines Gerätes oder Netzwerks sollte im Idealfall eindeutig sein.
 - Länge: 32 Bit

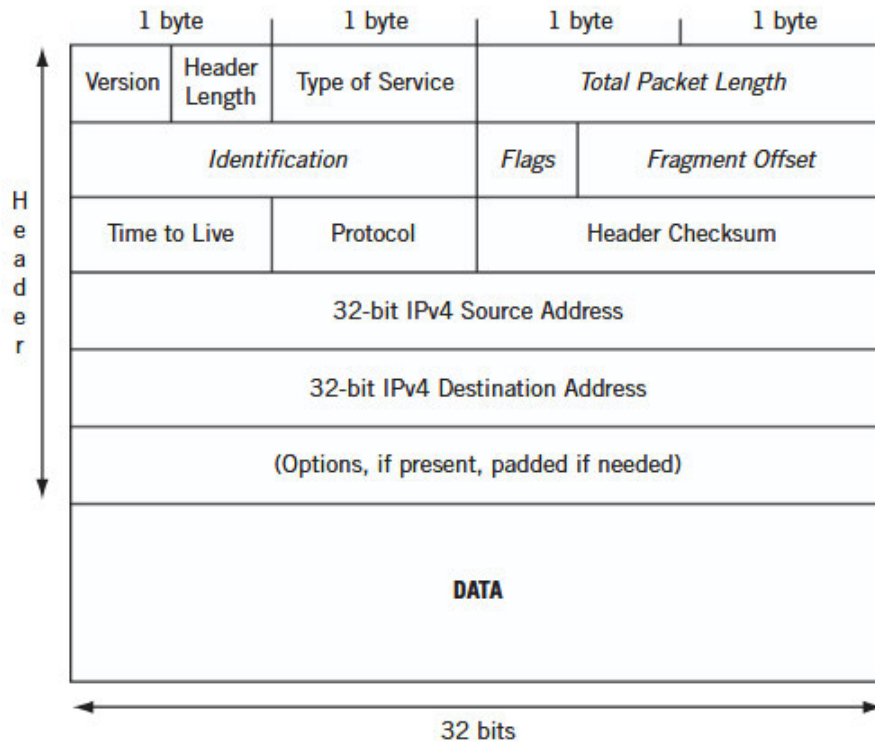


Abbildung 1: Aufbau des IP-Headers [Gor17, Abbildung 7.3]

6.2 Aufbau von KNX Telegrammen

KNX TP1 unterstützt drei verschiedene Telegrammformate [Ass21b, Abschnitt 2.2.1]: das Acknowledgement Frame, das L_Poll_Data Frame und das L_Data Frame.

6.2.1 KNX-Acknowledgements

Acknowledgements dienen dabei der gleichen Funktion wie in TCP und bestätigen den Empfang von Nachrichten [Ass21b, Abschnitt 2.2.7]. Allerdings unterscheiden sich die dabei verwendeten Implementationen stark voneinander. In TCP wird jedes erhaltene Paket durch ein Acknowledgement bestätigt. Dieses besteht aus einem vollen IP/TCP-Header und enthält die Sequenznummer des Paketes, das bestätigt werden soll. In KNX besteht das Acknowledgement Frame lediglich aus einem einzelnen Byte, welches die Antworten Ack, Nak, Busy und Nak+Busy kodieren kann. Die Zustellung zum

Absender funktioniert, da das Ack an alle Geräte des Busses übertragen wird. Da das Ack keine Sequenznummern verwendet, erfolgt die Zuordnung zum Telegramm, das bestätigt werden soll, über die zeitliche Abfolge. Nach dem Empfang eines Telegramms gibt es einen reservierten Zeitraum, in dem das Acknowledgement erwartet wird [Ass21b, Abschnitt 2.4.2].

Damit das Acknowledgement korrekt auf ein TCP Paket abgebildet werden kann, müsste also ein Zwischenspeicher geschaffen werden, welcher die zu bestätigenden Pakete aufbewahrt, damit aus diesen relevante Daten für die Zustellung in IP, wie Quell- und Zieladresse ausgelesen werden könnten. Bei Empfang eines KNX-Acknowledgements müsste dann ein entsprechendes TCP-Ack erstellt und gesendet werden. Bei nicht erhaltenen Acknowledgements müssten darüber hinaus Time-Outs erstellt werden, um ältere Pakete zu verwerfen. Zudem wäre zu erwarten, dass bei hoher Netzlast Race-Conditions zu fehlerhaften Acks führen könnten, welche ebenfalls behandelt werden müssten. Da das KNX-Acknowledgement nur ein Byte groß ist und in seinem Aufbau nur sehr geringe Spielräume zulässt, scheint es unwahrscheinlich, dass sie als Angriffsvektor genutzt werden können. Daher wird die Abbildung von Acknowledgements nicht umgesetzt. Sie wird allerdings im Ausblick behandelt.

6.2.2 L_Poll_Data

Das L_Poll_Data Frame wird von einem sogenannten Poll_Data Master an eine Poll Gruppe gesendet. Die Geräte, die einer Poll Gruppe angehören (Poll Slaves), antworten, ähnlich wie schon beim Acknowledgement, in definierten Zeitslots auf die Anfrage [Ass21b, Abschnitt 2.2.6.1]. Dabei ist für jeden Slave ein spezifischer Zeitslot vorgesehen. Das Mapping eines vollständigen L_Poll_Data Requests steht also vor den gleichen Schwierigkeiten wie das Acknowledgement, da in den Antworten (Poll_Data characters) auf ein Poll_Data Frame nicht steht, welches Gerät die Antwort gesendet hat oder auf welche Anfrage geantwortet wird [Ass21b, Abschnitt 2.2.6.1]. Diese Information folgt aus dem Kontext (dem definierten Antwortzeitslot des Slaves [Ass21b, Abschnitt 2.4.3]) und wird daher nicht explizit im Poll_Data Charakter übertragen. Da potenziell mehrere Slaves nacheinander auf ein L_Poll_Data Frame antworten, erfordert die Zuordnung der Antworten zu spezifischen Slaves demnach Wissen über Daten, die nicht im Netzwerk übertragen werden. Wie das Acknowledgement besitzt auch das L_Poll_Data Frame keine APCI [Ass21b, Abschnitt 2.2.6.1], also keine Instruktionen und Payload, womit es unwahrscheinlich ist, dass es für Angriffe genutzt werden kann. Deshalb wird auch das L_Poll_Data Frame nur im Ausblick behandelt.

6.2.3 L_Data_Frame

Das L_Data_Frame Format sieht zwei mögliche Varianten vor. Das L_Data_Standard_Frame und das L_Data_Extended_Frame. Wie die Namen errahnen lassen, entscheidet die Länge der gesendeten Daten, welches Frame genutzt wird.

- **Quelladresse und Zieladresse:**
 - Beschreibung: Die von KNX verwendeten Geräteadressen haben eine Länge von zwei Byte. Davon kodieren die ersten vier Bit die **Area**, die zweiten vier die **Line** und weitere 8 Bit die Adresse des spezifischen Geräts. Dieses Adressschema bietet insgesamt 65536 Geräten Platz. Damit sind die Adressen halb so lang wie die in IPv4 verwendeten.
 - Länge: je 16 Bit
- **Hop-Count:**
 - Beschreibung: Der Hop-Count funktioniert entsprechend zur TTL in IP. Sie wird bei jeder Weiterleitung um eins reduziert, erreicht sie Null wird das Telegramm verworfen. Da KNX durch seinen Aufbau garantiert, dass keine Zyklen bei der Übertragung von Telegrammen auftreten, dient der Hop Count hauptsächlich dazu, Pakete auf eine bestimmte Reichweite zu beschränken.
 - Länge: 3 Bit
- **Länge:**
 - Beschreibung: Die Länge gibt die Gesamtlänge des Telegramms an.
 - Länge: 4 Bit
- **TPCI:**
 - Beschreibung: Die TPCI enthält Informationen darüber, welche Art von Kommunikation verwendet wird.
 - Länge: 7 Bit, aufgeteilt in ein Einzelnes und einen 6 Bit Block
- **APCI:**
 - Beschreibung: Die APCI definiert, welche Operation mit den Daten im Payload ausgeführt werden soll. Dies Ähnelt in der Funktion zwar dem Protokollfeld des IP Headers oder der DSCP, da die Felder jedoch in beiden Standards konkrete Anweisungen definieren, ist ein Mapping der APCI in eines der beiden anderen Felder vermutlich eher irreführend als hilfreich.
 - Länge: Variabel
- **Prüfsumme:**
 - Beschreibung: Am Ende des Telegramms befindet sich außerdem eine Prüfsumme, mit welcher die korrekte Übertragung von Paketen sichergestellt werden kann. Hierfür verwendet KNX ein bitweises logisches NOT XOR [Ass21b, Abschnitt 2.2.4.6].
 - Länge: 8 Bit

6.2.5 L_Data_Extended

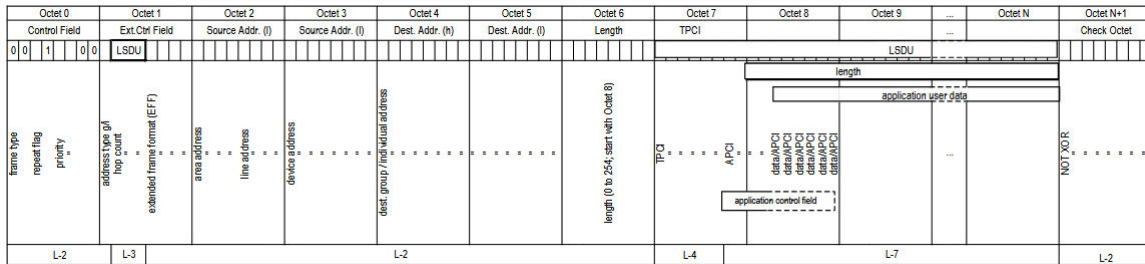


Abbildung 3: Aufbau des Data_Extended Frames [Ass21b, Abbildung 33, Abschnitt 2.2.5.1]

Das L_Data_Extended Frame ist sehr ähnlich aufgebaut wie das L_Data_Standard Frame und besitzt die gleichen Felder. Allerdings ist das Längenfeld 8 Bit lang (also doppelt so groß, wie beim Standardformat), damit die Länge von großen Paketen abgebildet werden kann. Dadurch müssen der Hop Count und das separate Bit der TPCI (der Address Type), welche beim Standardframe im gleichen Byte wie die Länge übertragen werden, in ein anderes Byte ausgelagert werden, das sogenannte erweiterte Kontrollfeld. Das erweiterte Kontrollfeld enthält außerdem das Extended Frame Format, welches zwischen Point-to-Point- und Broadcastkommunikation unterscheidet (vgl. [Ass21b, Abschnitt 2.2.6]).

6.3 cEMI Frame

Für die Übertragung von Telegrammen über Ethernet sieht der KNX Standard KNXnet/IP als Standardprotokoll vor [Ass21c, Abschnitt 1.5]. Die gesendeten Telegramme sollen außerdem im common External Message Interface (cEMI) Format versendet werden [Ass21c, Abschnitt 2.2]. Ein Telegramm wird somit vom KNXnet/IP Header und dem cEMI Header eingefasst und als Payload eines UDP Paketes über Ethernet versendet.

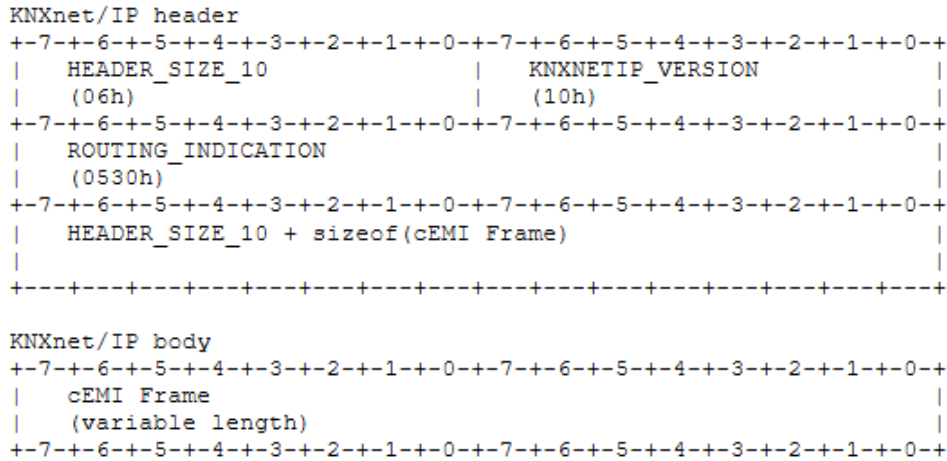


Abbildung 4: [Ass21a, Abbildung 1 in Abschnitt 4.1.2]

Wie in Abbildung 4 zu sehen ist, besteht der KNXnet/IP Header aus Parametern, die zur Weiterleitung erforderlich sind, die aber nicht innerhalb einer reinen KNX Installation genutzt werden. Im Mapping ist daher vor allem die Headerlänge relevant, um das cEMI Frame extrahieren zu können.

Message Code	Additional Info Length	Additional Information	Control field 1	Control field 2	Src. High	Src. Low	Dest. High	Dest. Low	NPDU	
MC	AddIL	...	Ctrl1	Ctrl2	SAH	SAL	DAH	DAL	L	TPCI/APCI & data
29h	x0r0ppxx

Abbildung 5: Aufbau eines cEMI Frames [Ass21c, Abschnitt 4.1.2, Abbildung 2]

Das cEMI Frame selbst ist dabei eine generische Struktur, welche Formate aller KNX Medien unterstützen kann (vgl. [Ass21d] Abschnitt 4.1.1). Es verpackt das Telegramm in einen Header, welcher aus zwei Pflichtfeldern besteht, die jeweils ein Byte lang sind. Diese sind der Messagecode für die Kennzeichnung des genutzten Formates sowie ein Feld für die "zusätzliche Headerlänge", welche angibt, wie viele weitere Bytes auf die Pflichtfelder folgen. Wie in der Abbildung gezeigt ist, entspricht das so verpackte Telegramm dem L.Data.Extended Format. Im Mapping wird das L.Data Frame extrahiert und äquivalent zu den klassischen L.Data Frames gemappt.

6.4 Implikationen für das Mapping

Sowohl die IP-Pakete als auch die L.Data Frames besitzen Felder, die spezifisch für ihr jeweiliges Medium sind und die sich nur schwer miteinander vergleichen lassen. Anders als beim KNX-Acknowledgement oder dem L.Poll Frame sind die grundlegenden Funktionen jedoch vergleichbar.

Beide Standards nutzen individuelle Geräte- oder Netzwerkadressen, um den Zielort ihrer Pakete anzugeben. Insbesondere verwendet IPv4 längere Adressen als KNX, sodass

Quell- und Zieladresse eines Telegramms vollständig in die entsprechenden Felder des IP-Headers übertragen werden können. Die übrigen 2 Byte werden dabei von einer Netzwerkadresse ausgefüllt. Dadurch wird den Paketen ein dediziertes Netzwerk zugeordnet, welches von einem IDS überwacht werden kann. Durch diese Übertragung erhält zudem jeder Kommunikationskanal eine eigene und konstante IP-Adresse, sodass eine Zuordnung zu Geräten, oder Gruppen auch über mehrere Nachrichten hinweg möglich bleibt. Beispielsweise wäre es möglich zu erkennen, dass ein Gerät untypisch viele Nachrichten versendet.

Der Hop-Count erfüllt zwar praktisch eine andere Aufgabe als die TTL in IP, aber da die beiden Felder logisch gleich behandelt werden, wird der Hop Count in das TTL Feld eingetragen.

Die Länge der KNX-Telegramme ist zwar ein relevanter Parameter, um sie korrekt empfangen zu können, aber da der IP Header eine unterschiedliche Länge besitzt, würde es zu einer fehlerhaften Übertragung führen deren Wert direkt zu übertragen. Stattdessen wird die vom IP Standard vorgesehene Länge verwendet.

Bei der Prüfsumme verhält es sich ähnlich. KNX und IP benutzen zwar beide ein Verfahren, um zu prüfen, dass Telegramme korrekt angekommen sind, eine KNX-Prüfsumme in einem IP-Telegramm würde dieses aber lediglich ungültig machen. Daher soll vor dem Mapping geprüft werden, ob die Prüfsumme des KNX-Telegramms korrekt ist. Wenn dies der Fall ist, wird auch für das IP-Paket die korrekte Prüfsumme berechnet. Andernfalls wird die Prüfsumme mit einer falschen überschrieben, um zu markieren, dass ein Fehler bei der Übertragung aufgetreten ist.

KNX befolgt das 7 Schichten OSI-Modell. Dabei wird die Anwendungsschicht durch die APCI und den darin enthaltenen Payload repräsentiert. Im IP-Protokoll werden diese Informationen innerhalb des UDP-Headers gesendet, weshalb die APCI, inklusive der Nutzdaten, als Payload gemappt wird. Somit ist ein IDS auch in der Lage, den Payload der Telegramme mit seinen Regeln abzugleichen. In Snort erfolgt dies beispielsweise durch Payload Detection Rules [Tea]. All diese Beobachtungen gelten auch für das Mapping von cEMI Frames. Da die Telegramme für die Übertragung mittels Ethernet um den KNX/IP- sowie den cEMI-Header erweitert werden, muss das Telegramm, anders als beim Mapping des reinen L_Data Frames, zunächst extrahiert werden.

7 Übersicht

In diesem Abschnitt soll der Aufbau der verwendeten Komponenten beschrieben werden. Dazu gehören eine physische KNX-Installation, bestehend aus zwei Sensoren¹, einem Aktor² und einem KNX/IP-Router³, ein RaspberryPi 4 sowie eine JavaCard und ein Kartenlesegerät der Firma Reiner⁴. Auf dem RaspberriPi läuft die Mappingsoftware in Python v3.11.2, die API des sicheren Schlüsselspeichers in Java-21 sowie Snort3. Im

¹KNX IO 511.1 Secure (1021)

²BE-TA5508.01

³KNX IP Router 752 Secure

⁴BSI-K-TR-0067-2010

Folgenden wird zunächst der Aufbau der physischen Komponenten beschrieben und anschließend auf die Konfiguration und das Zusammenspiel der Software-Komponenten auf dem Raspberry Pi eingegangen.

7.1 Physische Komponenten

In der untenstehenden Grafik ist der schematische Aufbau der genutzten KNX Installation zu sehen. Über den KNX Bus sind zwei Sensoren und ein Aktor miteinander verbunden. Der dabei entstehende Netzwerkverkehr kann wahlweise im cEMI-Format über den KNX/IP-Router an den RaspberryPi weitergeleitet werden oder über eine USB-Bridge direkt vom Bus abgegriffen werden. Im zweiten Fall erhält man die Pakete ohne den IP-Header und im KNX TP1-Format. Zu jedem Zeitpunkt kann die Mappingsoftware aber nur eine der beiden Weiterleitungen nutzen. Der Raspberry Pi kann darüber hinaus einen sicheren Schlüsselspeicher ansprechen, um über diesen die MAC zu verifizieren und KNX-Data-Secure Telegramme zu entschlüsseln.

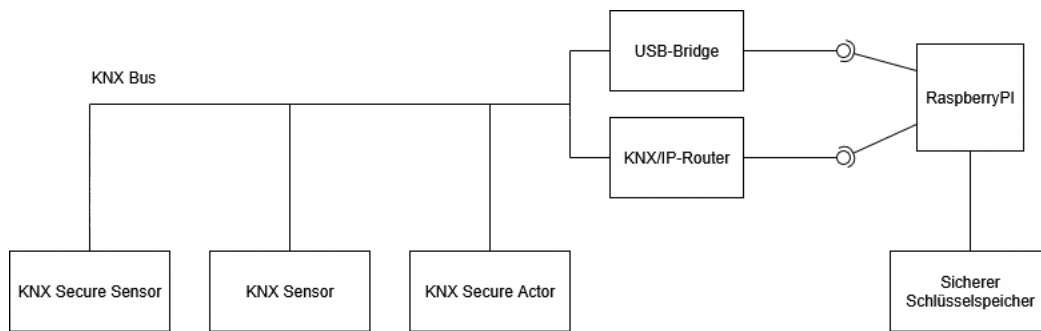


Abbildung 6: Darstellung der physischen Komponenten [Schb]

7.2 Software Komponenten

Der grobe Aufbau ist in Abbildung 7 zu sehen. Je nach Wahl des Input-Geräts werden Telegramme von der Mappingsoftware entweder über das Netzwerkinterface eth0 oder das Geräte-File der USB-Bridge empfangen. Falls die Telegramme verschlüsselt oder mit einem MAC versehen sind, werden sie an die API des sicheren Schlüsselspeichers weitergegeben, welcher sie wiederum zur Entschlüsselung und MAC-Verifikation an das Java-Card Applet weiterreicht. Anschließend werden die entschlüsselten Daten über die API zurück an den Mapper geliefert.

Der Mapper selbst kreiert zunächst ein virtuelles Netzwerkinterface auf dem Raspberry Pi, genauer ein Tun-Interface. Anschließend nutzt er die erhaltenen KNX-Telegramme, um daraus inhaltlich möglichst ähnliche IP-Pakete zu konstruieren. Diese werden dann an das Tun Gerät gesendet. Ein IDS, in dieser Arbeit Snort3, überwacht das geschaffene Tun Interface und erhält darüber indirekt Zugriff auf die gemappten KNX-Telegramme.

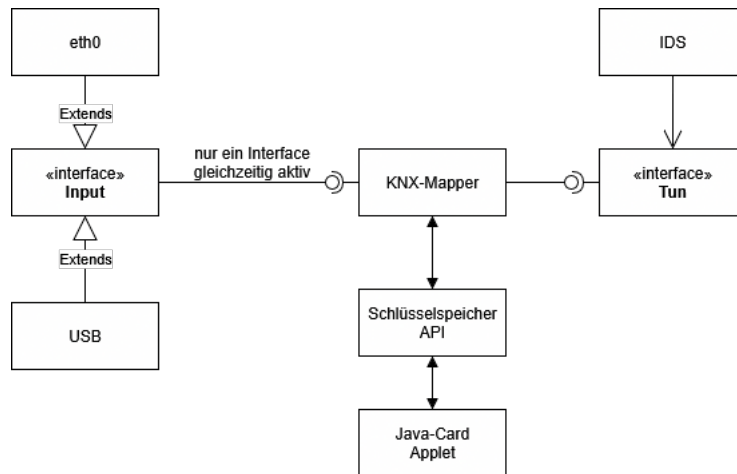


Abbildung 7: Darstellung der Softwarekomponenten [Schc]

8 Aufsetzen der Komponenten

Damit die Telegramme vom KNX-Netzwerk gemappt und an ein IDS übertragen werden können, war es nötig, eine Reihe von Treibern und Applikationen zu installieren. Damit dieser Prozess nachvollziehbar bleibt, sollen die Abhängigkeiten in diesem Abschnitt beschrieben werden.

8.1 KNX-Installation

Um das Mapping mit allen Komponenten testen zu können, wurde eine KNX Installation aufgebaut. Für die Konfiguration der Geräte wurde das ETS-Tool verwendet [Assc], welches von der KNX-Association zu diesem Zweck entwickelt wurde. Das genutzte Setup ist in Sektion 7.1 genauer beschrieben.

Für die Nutzung des KNX-Mappers besteht darüber hinaus die Notwendigkeit, die Telegramme an den RaspberryPi zu übertragen. Hierfür wurden zwei Ansätze verwendet:

- *USB-Bridge*: Eine USB-Bridge wird direkt in den Bus eingebunden und übermittelt alle Telegramme ohne weitere Header. Wird dieser Ansatz verwendet, entsprechen die empfangenen Telegramme dem im Standard vorgesehenen Format für TP1. Üblich sind dabei die L_Data Frames ([Ass21b] Abschnitt 2.2.1).
- *KNX/IP-Router*: Der KNX/IP-Router wird als weitere Komponente an das KNX-Netzwerk angeschlossen. Gemäß der KNX IP-Spezifikation ([Ass21a] Abschnitt 4.2.1) werden die Telegramme des KNX-Netzwerks um einen KNX/IP-Header ergänzt und als Teil des IP/UDP-Protokollstapels im cEMI-Format übertragen.

8.2 Reiner SCT cyberJack Kartenleser BSI-K-TR-0067-2010

Für den Zugriff auf die Javacard wurde ein Chipkartenleser der Firma Reiner genutzt. Anders als andere Kartenleser benötigt dieser einen Treiber, um vom Betriebssystem erkannt zu werden [Co].

Zusätzlich benötigt Java die `libpcsclite1` Bibliothek, um mit dem Kartenleser interagieren zu können [Deb].

Auch kann es vorkommen, dass das Betriebssystem den Zugriff auf den Kartenleser verweigert. Die entsprechenden Berechtigungen können über die Kommandozeile gesetzt werden.

8.3 Mappingsoftware

Die Mappingsoftware kann unter <https://scm.cms.hu-berlin.de/tom.schuett/KNXtoIP> [Scha] heruntergeladen werden. Das Skript besitzt einige Abhängigkeiten, die zunächst erfüllt sein müssen, bevor es ausgeführt werden kann. Dazu gehören die genutzten Standardbibliotheken für Python 3. Diese sind in der `.venv` Datei aufgeführt und können mittels `pip install .venv` installiert werden.

Außerdem gibt es einige Parameter, die nicht statisch vergeben werden können, da sie sich zwischen Geräten und je nach Anwendungsfall unterscheiden. Für die Konfiguration wird daher eine `config.ini` Datei genutzt, welche auf die jeweilige Umgebung angepasst werden muss. Die einzelnen Parameter der config werden in Abschnitt 10.1 näher erläutert. Im Gitprojekt der Mappingsoftware ist die config zudem beispielhaft ausgefüllt.

Da die Mappingsoftware auf Netzwerkschnittstellen zugreift, muss die Anwendung mit den hierfür notwendigen Berechtigungen vom Betriebssystem gestartet werden. Unter Unix-Systemen können diese in der Kommandozeile durch den Präfix `sudo` temporär vergeben werden. Alternativ können auch die Berechtigungen des entsprechenden Nutzers erweitert werden.

8.4 Java API

Eine genaue Beschreibung, wie der sichere Schlüsselspeicher aufgesetzt werden kann, findet sich in der Bachelorarbeit von Vincent Weigt [Wei24]. Daher soll an dieser Stelle nur ergänzt werden, dass die API zur Kommunikation mit dem Kartenleser die `libpcsclite1` Bibliothek benötigt [Deb], welche über den Paketmanager installiert werden kann.

Sobald alle diese Schritte ausgeführt wurden, sollte man beliebige cEMI-Pakete sowie die KNX-Telegramme der unterstützten Typen an den ausgewählten Port senden können, von wo aus sie gemappt und an den Output-Port weitergeleitet werden. Die erfolgreiche Installation kann durch Ausführen der `KeystoreTest` Klasse verifiziert werden.

8.5 Snort3

Das Team der Cisco Gruppe hat eine ausführliche Dokumentation für die Installation von Snort geschrieben. Diese ist unter <https://docs.snort.org/start/installation> zu finden.

Nachdem Snort erfolgreich installiert wurde, muss es noch für das Projekt konfiguriert werden. Hierfür ist zunächst wichtig, dass Mappingsoftware bereits installiert ist und mindestens einmal mit dem gewünschten Interface gestartet wurde. Andernfalls steht kein Tun Interface bereit, das Snort überwachen kann.

Für die Konfiguration von Snort wird eine `.lua` Datei verwendet. In dieser kann unter anderem eingestellt werden, welche Regeln Snort für die Erkennung von Gefahren nutzt und in welcher Datei Alerts gespeichert werden sollen.

Die Regeln selbst werden in einer `.rules` Datei geschrieben, welche dann in der `.lua` angegeben wird. Mit dem Befehl

```
sudo /path/to/snort/snort -c /path/to/snort/snort/etc/snort/snort.lua
```

können die von der Mappingsoftware generierten Pakete erfolgreich geloggt werden. `sudo` ist notwendig, da Snort Adminrechte benötigt, um auf das Tun Interface zuzugreifen. Dahinter wird der Pfad zu Snort definiert. Der Parameter `-c` spezifiziert, aus welcher `.lua` Datei die Konfiguration übernommen werden soll.

Darüber hinaus gibt es noch weitere Parameter, welche zur Konfiguration genutzt werden können.

- `-i` spezifiziert das zu überwachende Interface.
- `-l` gibt den Pfad an, an dem das erstellte Log abgelegt wird.
- `-A` definiert, in welchem Format die Alerts ausgegeben werden sollen.

Beim Testen des Mappers wurde der Befehl `sudo /path/to/snort/snort -i [Name des Interface] -c /path/to/snort/snort/etc/snort/snort.lua -l /folder/for/log/` genutzt.

9 Aufbau der Mapping Software

Wie bereits in Abschnitt 5.2 besprochen, erhält die KNX-Mapping-Software als Input KNX-Telegramme entweder über Ethernet oder eine USB-Verbindung und erstellt und sendet anschließend die gemappten Pakete an ein TUN Interface.

Dieses Kapitel betrachtet, welche Komponenten intern existieren, wie diese zusammenhängen und welche Designentscheidungen dabei getroffen wurden.

Die zentralen Python Module sind:

- `config.ini`: enthält Konfigurationsparameter.
- `KNX-Telegramme.py`: enthält Datentypen, welche die KNX-Telegramme abbilden, sowie eine Funktion, um sie in IP umzuwandeln.
- `KNXasIPFactory.py`: enthält Funktionen, welche auf ein KNX-Interface aufgesetzt werden können und von dort Pakete empfangen und IP-Pakete zurückgeben.
- `keystore_manager.py`: erstellt und verwaltet eine JVM, inklusive ihrer Funktionen um Zugriff auf die API des sicheren Schlüsselspeichers zu ermöglichen.

- `errors.py`: enthält Exceptions, die geworfen werden können für besseres Logging.
- `TunTap.py`: enthält Funktionen, um ein Tun-Interface zu starten.
- `util/util_general.py`: enthält allgemein nützliche Funktionen, die von anderen Klassen verwendet werden.
- `util/util_config.py`: enthält Funktionen, um die Korrektheit der `config.ini` sicherzustellen und korrekt zu formatieren.
- `main.py`: ist die main-Funktion, welche die Zugriffe auf die anderen Funktionen koordiniert.

Das Projekt enthält noch weitere Komponenten, diese werden aber während des Mappings, wie es im Labor getestet wurde, nicht verwendet, sondern dienen lediglich Test- und Entwicklungszwecken. Sie werden in Kapitel 10.10 beschrieben.

10 Erklärung der Module

10.1 `config.ini`

In der `config.ini` werden die nötigen Einstellungen getroffen, um die Mappingsoftware auf ihre Einsatzumgebung einzustellen.

Diese sind:

- Der `Mapper`, welcher die verwendete Schnittstelle zum KNX-Netzwerk angibt (USB oder Ethernet).
- Der `inPort`, welcher entweder das Ethernet-Interface oder die Geräte-Datei der USB-Bridge enthält.
- Der `outPort`, welcher den Adressbereich des erstellten Tun-Interfaces definiert. Es ist notwendig, ein Netzwerkinterface der Klasse B oder größer aufzusetzen.
- Die `Java-API`, welche den Dateipfad zur Java-API enthält.
- Die `test_tun_net_address`, welche die Netzwerkadresse des Tun Gerätes, dass beim Testen zum Paketempfang genutzt wird, kodiert.
- Der `verbose`-Parameter, welcher angibt, wie detailliert das System über Fehler und Warnungen Auskunft gibt.

10.2 keystore_manager

Dieses Modul bündelt alle Aufrufe an die Java-API. Dabei wird die `jpye`-Bibliothek genutzt, um eine Java Virtual Machine (JVM) zu starten und die Funktionen der API aufzurufen. Da das Starten einer JVM mit einem hohen Ressourcenaufwand verbunden ist, muss sichergestellt sein, dass nur eine JVM zusammen mit der Mappingsoftware gestartet wird und alle Zugriffe auf die API von dieser übernommen werden.

Die enthaltenen Funktionen sind:

- `ensure_jvm_up`: Die Funktion prüft, ob bereits eine JVM hochgefahren wurde und startet anderweitig eine JVM. Anschließend gibt sie den Status der JVM (bei korrektem Ablauf `True`) zurück. Alle Funktionen, welche direkt auf die Java-API zugreifen, werden aus diesem Modul gestartet und nutzen somit ebenjene JVM.
- `get_keystore`: Die Zugriffe auf die Java-Card werden durch einen sogenannten keystore koordiniert. Dieser muss auf der Java-Card initialisiert werden, dies darf aber nur ein mal geschehen. Diese Funktion prüft daher, ob der keystore bereits erstellt wurde und erstellt und initialisiert im Negativfall einen keystore. Anschließend wird der keystore zurückgegeben.
- `get_exception`: Diese Funktion gibt eine Exception zurück, welche genutzt wird, um eine fehlgeschlagene MAC-Prüfung zu erkennen. Da die Exception von der Java-API geworfen wird, muss diese über die JVM importiert werden, weshalb sie nicht im `errors.py` Modul zu finden ist.

10.3 KNX_Telegramme

Dieses Modul enthält vor allem zwei Klassen `KNX_TP1_Telegramm` und `KNX_IP_cEMI_Frame`. Beide Klassen dienen dazu, die Datenfelder eines KNX Telegramms zu bündeln und die Funktion `as_IP` bereitzustellen, welche ein gemapptes IP-Paket zurückgibt.

`KNX_TP1_Telegramm` wird genutzt, um Telegramme von der USB-Bridge zu mappen, `KNX_IP_cEMI_Frame` hingegen für Telegramme über den KNX/IP Router.

Hierfür besitzen beide Klassen eine `init` Funktion, welche das ursprüngliche KNX-Telegramm als Bytearray übergeben bekommt und daraus ein Objekt mit einzeln adressierbaren Feldern wie dem Kontrollfeld oder der Zieladresse erstellt. Hierfür nutzen sie `get` Funktionen, welche die entsprechenden Bereiche aus dem Bytearray extrahieren.

Die Klassen besitzen darüber hinaus eine Funktion, welche von außen aufgerufen werden kann und welche das Paket gemappt auf IP zurückgibt. Je nach Medium, von dem die Telegramme empfangen werden, besitzen die Klassen darüber hinaus noch spezielle Funktionen.

10.3.1 KNX_TP1_Telegramm

Betrachten wir zunächst `KNX_TP1_Telegramm`.

Es besitzt Variablen für die folgenden Datenfelder:

- `telegram_as_byte_array`: Das gesamte Telegramm
- `type_of_telegram`: `L_Data_Standard` oder `L_Data_Extended`
- `checksum`
- `data_len`
- `APDU`
- `src`
- `dst`
- `hop_count`
- `control1`

Die Funktion, welche das Telegramm auf ein IP-Paket mapped, ist `as_IP`. Sie setzt die in Abschnitt 6.4 überlegten Implikationen für das Mapping um. Da sie ein zentraler Teil des Mappings ist, wird ihre Funktionsweise im Folgenden genauer erklärt.

Zur Erstellung der IP-Nachricht wird die Bibliothek Scapy verwendet [Com]. Diese ermöglicht das erstellen von syntaktisch korrekten IP/UDP Paketen und berechnet die korrekte Länge und Prüfsumme. Falls ein KNX Telegramm mit fehlerhafter Prüfsumme empfangen wird, wird die Prüfsumme des erstellten IP-Paketes nachträglich verfälscht, indem sie auf `0xFFFF` gesetzt wird. Sollte die Prüfsumme bereits diesem Wert entsprechen, wird stattdessen `0x0000` verwendet. Da die Geräteadressen von KNX halb so lang sind wie die von IP, wird die Funktion `assemble_address` genutzt, um die aus der KNX-Geräteadresse und der Netzwerkadresse des `outPort` eine zum Gerät eindeutige IP-Adresse zu konstruieren. Die ersten 16 Bit entsprechen dabei der Netzwerkadresse, während die zweiten 16 die ursprünglichen KNX-Geräteadresse enthalten. Auf diese Weise stellen wir sicher, dass jede IP-Adresse genau einem KNX-Gerät zugeordnet ist. Das IDS sollte also nachvollziehen können, welche Geräte miteinander kommunizieren und es wäre auch möglich, Regeln zu erstellen, die sich an der Struktur des KNX-Netzwerkes orientieren. Ein Beispiel hierfür wäre sinngemäß: „Nur Geräte, die auf dem gleichen Bus liegen, sollen Nachrichten austauschen können“. Für dieses Beispiel müsste lediglich überprüft werden, dass die Line-Address des Senders mit der des Empfängers übereinstimmt. Der Hop-Count wird, wie in Kapitel 6 erklärt, in das Feld der TTL geschrieben.

Zu guter Letzt wird die gesamte APDU, also der Bestandteil des KNX-Telegrams, welcher Daten oder Anweisungen für den Empfänger enthält, als UDP-Payload in die Nachricht eingesetzt. Damit die APDU mit bekannten Angriffen abgeglichen werden kann, ist es wichtig, dass dieser Bereich unverändert übertragen wird und insbesondere

auch nicht fragmentiert. Wie in der KNX Spezifikation vermerkt ist, beträgt die maximale Länge eines KNX Frames 254 Bytes Payload sowie 8 Bytes im Header und ein weiteres für die Prüfsumme [Ass21b, Abschnitt 2.2.5.1, Abbildung 33]. Dies ist kleiner, als die maximale unfragmentiert übertragbare Länge, sowohl in UDP als auch IP. Es ist daher nicht zu erwarten, dass es hierbei zu Problemen kommt. Sicherheitshalber wird trotzdem das Do not Fragment Flag gesetzt.

10.3.2 KNX_IP_cEMI_Frame

Das KNX_IP_cEMI_Frame entspricht in seinem Aufbau dem L_Data_Extended Format, allerdings ergänzt um den cEMI sowie KNX IP Header. Das Mapping verläuft dementsprechend ähnlich. Die Informationen aus den Headern werden für das Mapping nicht benötigt und werden entsprechend entfernt. Die Datenfelder sind prinzipiell auch die gleichen. Allerdings muss berücksichtigt werden, dass das cEMI Format sowohl unverschlüsselte als auch verschlüsselte Telegramme unterstützen soll. Dementsprechend muss zunächst unterschieden werden, ob eine Entschlüsselung notwendig ist. Die entsprechende Information befindet sich in der APCI, welche wiederum ein Teil der APDU ist. Deshalb muss die APDU in die APCI und die Nutzdaten zerlegt werden, bevor sie gemappt werden kann.

Ein weiterer Unterschied besteht darin, dass verschlüsselte Telegramme keine Prüfsumme verwenden, sondern die korrekte Übertragung stattdessen über einen MAC sicherstellen. Wenn der sichere Schlüsselspeicher Daten entschlüsselt nimmt er zugleich eine Überprüfung des MACs vor. Falls die MAC Prüfung fehlschlägt, wird eine entsprechende Fehlermeldung, jedoch kein entschlüsseltes Payload ausgegeben. In diesem Fall wird die Prüfsumme des IP-Paketes ebenfalls verfälscht, die Nutzdaten müssen durch einen zweiten Aufruf des Schlüsselspeichers entschlüsselt werden, welcher keine Überprüfung der MAC vornimmt. Das Mapping der restlichen Datenfelder findet äquivalent zum KNX_TP1_Telegramm statt.

Durch dieses Mapping sollten die für die Überwachung relevanten Eigenschaften der beiden Telegrammformate erhalten bleiben und eine zielführende Überwachung ermöglicht werden. Wie in Kapitel 6 bereits angesprochen wurden einige Datenfelder, wie etwa das Repeatflag, nicht übertragen. Es wird aber davon ausgegangen, dass dies die Gefahrenerkennung nur geringfügig verschlechtert.

10.4 util_general

Das Modul enthält eine Sammlung von Helferfunktionen, die im Folgenden aufgelistet werden.

- `is_L_Data_Standard_Frame`, `is_L_Data_Extended_Frame`, `is_L_Poll_Data_Frame`, `is_Ack_Frame`: Alle diese Funktionen erhalten ein Telegramm als Bytearray und geben True zurück, wenn das Telegramm dem Format der Funktion entspricht. Im aktiven Mapping werden nur Überprüfungen für das Standard und Extended Frame genutzt, da die anderen Formate nicht gemappt werden.

- `catch_eth`: Diese Funktion wird genutzt, wenn die Telegramme über einen KNX/IP Router erhalten werden. Die Funktion erhält den Namen eines Ethernet-Interfaces als Parameter und nutzt die `sniff` Funktion von Scapy, um eine neue Nachricht zurückzugeben, falls eine seit dem letzten Aufruf empfangen wurde.
- `obtain_payload`: Diese Funktion wird genutzt, um das KNX-Telegramm aus einem Ethernet Frame zu extrahieren und sicherzustellen, dass es sich um ein KNX-Telegramm handelt. Hierfür testet sie zunächst, dass es sich bei der Eingabe um ein Packet der Scapy-Bibliothek handelt und dieses ein UDP-Payload besitzt. Außerdem stellt sie sicher, dass UDP den Port 3671 verwendet, welcher für die Nutzung von cEMI vorgesehen ist [Ass21a, Abschnitt 2.1]. Außerdem stellt sie noch sicher, dass das Packet ein Payload enthält. Wenn alle Tests erfolgreich verlaufen, gibt die Funktion das extrahierte Telegramm als Bytearray zurück. Andernfalls informiert sie über eine entsprechende Fehlermeldung darüber, dass die Eingabe kein KNX-Telegramm enthielt.
- `choose_mapper`: Die Funktion erhält von der `main` Funktion den eingetragenen Mapper aus der `config`. Entsprechend der Einstellung gibt die Funktion entweder den Mapper für die über USB erhaltenen KNX-TP1-Telegramme oder den Mapper für die über Ethernet erhaltenen cEMI-Telegramme zurück.

10.5 util_config

Das Modul `util_config` enthält Funktionen zur Nutzung der `config.ini`.

- `confirm_network_mask`: Die Funktion prüft, dass die Netzwerkmaske des `outPorts` kleiner gleich 16 Bit ist. Dies ist wichtig, da bei KNX-Installationen maximaler Größe zwei volle Bytes für die Geräteadresse benötigt werden. Falls das Netzwerk zu klein ist, bricht das Programm beim Starten mit einer entsprechenden Fehlermeldung ab.
- `assemble_java_path_for_jpype`: Hilfsfunktion, um den Pfad zur Java API des sicheren Schlüsselspeichers korrekt anzusprechen.

10.6 errors

Dieses Modul enthält eine Sammlung von Exceptions, die für alle anderen Module verwendet werden. Der `verbose` Parameter aus der `config` kann verwendet werden, um einzustellen, welches Level von Fehlern angezeigt wird. Dabei wird eine Skala von 0 bis 4 verwendet, mit den zugehörigen Gewichtungen `Critical`, `Error`, `Warning`, `Info` und `Debug`.

- **Critical:**
 - `notAMapperError`: wird ausgegeben, falls der in der config eingetragene Mapper nicht zugeordnet werden kann.
 - `networkToSmallError`: Die Exception welche von `confirm_network_mask` geworfen wird, wenn die Subnetzmaske falsch gesetzt ist.
- **Error:** Reserviert für zukünftige Erweiterungen, aktuell benötigt keine der Exceptions eine so schwere Fehlermeldung.
- **Warning:** Enthält zwei Warnungen, welche für Fehler im Telegrammaufbau gedacht sind. Aktuell werden diese aber nicht benutzt.
- **Info:** Die folgenden Fehler werden geworfen, während der Netzwerkverkehr darauf untersucht wird, ob es sich dabei um KNX-Telegramme handelt:
 - `notAPacketError`
 - `payloadIsEmptyError`
 - `noRawDataFoundError`
 - `noUDPFoundError`
 - `wrongUDPPortError`

Außerdem existiert noch der `telegramTypeNotSupportedError`, welcher für KNX-Telegramme geworfen wird, die nicht vom Mapper unterstützt werden.

- **Debug:** Dieses Level enthält keine Exceptions. Stattdessen werden Funktionen verwendet, welche die ihnen zugeordneten Informationen ausgeben, falls das Verbose-Level auf 4 gesetzt ist. Dazu gehören unter anderem das Ausgeben von Telegrammen oder Telegrammbestandteilen oder den Erhalt einer neuen Nachricht.

10.7 TunTap

Dieses Modul enthält die Funktion `start_up_tun`. Sie erhält den Namen für ein Interface, sowie dessen Netzwerkadresse samt Netzmaske als Eingabe.

Anschließend nutzt sie Kommandozeilenbefehle, um ein Tun-Interface mit dem gewünschten Namen und der gewünschten Netzwerkadresse zu erstellen. Diese kann dann sowohl als Eingang für Testzwecke, oder als Schnittstelle zum IDS dienen. Falls das Interface schon existiert, oder die Netzwerkadresse geändert wird, gibt die Funktion einen entsprechenden Vermerk in der Konsole aus und fängt die vom Betriebssystem geworfenen Exceptions. Sobald sichergestellt ist, dass das Interface in der gewünschten Konfiguration existiert, wird es außerdem direkt in den aktiven Zustand versetzt.

Da diese Funktionalität stark vom genutzten Betriebssystem abhängt, ist die Mappingsoftware somit nicht portabel. Es wird zudem davon ausgegangen, dass Adminrechte über `sudo` vorhanden sind.

10.8 KNXasIPFactory

Dieses Modul enthält die beiden zentralen Mappingfunktionen.

10.8.1 `map_incoming_traffic_from_eth`

Diese Funktion erstellt eine Queue und zwei Threads, die mit ihr interagieren. Der erste davon überprüft fortlaufend ein Ethernetinterface auf neue Telegramme und reiht diese in die Queue ein. Hierfür kommt die Funktion `obtain_payload` zum Einsatz. Der zweite Thread nutzt das Modul `KNX_Telegramme`, sowie die `send` Funktion um fortlaufend Telegramme aus der Queue zu mappen und an ein anderes Interface zu versenden.

Als Parameter benötigt die Funktion das überwachte Interface, sowie die Netzwerkadresse, an welche die Pakete gesendet werden sollen. Sie wird in Kombination mit dem KNX-IP-Router verwendet, oder beim Testen, um ein TUN-Interface zu überwachen.

10.8.2 `map_incoming_traffic_from_USB`

Diese Funktion funktioniert sehr ähnlich zu `map_incoming_traffic_from_eth`, empfängt ihre Pakete allerdings über eine USB-Schnittstelle. Sie ist für das Mapping von Telegrammen gedacht, welche über eine USB-Bridge erhalten werden.

10.9 `main`

Die `main`-Funktion ist das Herzstück der Mappingsoftware und die Funktion, die zu dessen Ausführung aufgerufen wird. Sie führt nacheinander die folgenden Schritte unter Zuhilfenahme der oben beschriebenen Module aus:

1. Starten des `IDS_tun`-Interfaces.
2. [Bei Nutzung des Testmodus] Starten des Testinterfaces.
3. Auswahl der korrekten Mappingfunktion (siehe `KNXasIPFactory`) und Start derselben in einem eigenen Thread.
4. Start der Receiverfunktion. Diese wird benötigt, um die gemappten Pakete aus dem `IDS_tun`-Interface zu empfangen. Üblicherweise werden Pakete an ein Interface gesendet und dort von einer Applikation gelesen. Da die Pakete der Mappingsoftware nicht an ein anderes Gerät oder einen anderen Empfänger gesendet werden, wird diese Aufgabe von der Receiverfunktion übernommen. Wenn diese Funktion nicht genutzt wird, werden die gemappten Telegramme zwar an das Interface gesendet, verbleiben dort allerdings im Buffer, bis sie verworfen werden. Das IDS kann sie in diesem Fall nicht erfassen, da die Telegramme nie erfolgreich gesendet werden.

Es stehen zwei Receiverfunktionen zur Auswahl, von denen eine die Telegramme lediglich empfängt und verwirft und die andere die empfangenen Telegramme in der Konsole ausgibt.

10.10 testing

In diesem Modul befinden sich verschiedene Funktionen, welche genutzt werden können, um die Mappingsoftware auch ohne ein angebundenes KNX-Netzwerk oder mit künstlich generierten Daten testen zu können. Damit die Anwendung die Testfunktionen nutzt, muss in der config der Mapper *Test* gewählt werden. Bei Ausführung des Programms wird dann in der main Funktion ein zweites TUN-Interface gestartet, welches als Ersatz für die Ethernet- oder USB-Schnittstelle dient. Der Name und die Netzwerkmaske des Testinterfaces werden ebenfalls in der config eingetragen.

Das Modul selbst enthält die folgenden Funktionen:

- `boot_up_test_inport`: Die Funktion erhält einen Namen, sowie eine Netzwerkadresse als Eingabe und stellt sicher, dass ein Tun Interface mit den gegebenen Parametern aktiv ist und über einen ausreichend großen Adressbereich verfügt. Anschließend wird der Filedescriptor des Tungeräts zurückgegeben.
- `sending_message`: Dies ist eine Hilfsfunktion, die ein Paket an ein Interface sendet.
- `spammer`: Die Funktion wird im Testmodus in einem eigenen Thread aufgerufen und nutzt wiederholt die `sending_message`-Funktion.
- `show_KNX_traffic_over_USB`: Dies ist eine Hilfsfunktion, die bei Verwendung des USB-Interfaces genutzt werden kann, um erhaltene Nachrichten anzuzeigen.

Zusätzlich enthält das Modul 3 valide KNX-Telegramme, in den Formaten `cEMI`, `L.Data.Standard` und `L.Data.Extended`, welche als Testtraffic gesendet werden können.

11 Tests

Um die Funktionalität des Mappers sicherzustellen, wurden drei verschiedene Testläufe durchgeführt.

- Funktionstest über die USB Bridge
- Funktionstest über den KNX/IP Router inklusive Ansprache des sicheren Schlüsselspeichers
- Lasttest unter Verwendung des `testing` Moduls

Folgend sollen die dabei entstandenen Logs aufgeführt und kurz erläutert werden.

11.1 Funktionstest über die USB Bridge

Zunächst wurde die Mappingsoftware für die Nutzung des USB Schnittstelle eingestellt und anschließend gestartet. Snort wurde so konfiguriert, dass es alle empfangenen Telegramme loggt. Um sicherzustellen, dass die Telegramme dem gewünschten Format entsprechen, wurde zudem das `IDS_Tun` Interface durch Wireshark überwacht.

Anschließend wurden ein Schalter der KNX-Installation zweimal an und wieder ausgeschaltet. Dadurch sollten vier unverschlüsselte L_Data_Standard Telegramme gesendet werden, welche sich wiederholen. Entsprechend würden wir im Log von Snort vier Alerts erwarten, die sich ebenfalls wiederholen.

```
Snort Log
10/27-17:00:46.366386 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/27-17:00:47.789745 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/27-17:00:49.542277 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/27-17:00:59.789822 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
```

Wie man erkennen kann, entsprechen die Alerts unseren Erwartungen. Die dazugehörigen Wireshark Logs können unter https://scm.cms.hu-berlin.de/tom.schuett/KNXtoIP/-/tree/main/Wireshark_Logs eingesehen werden.

11.2 Funktionstest über den KNX/IP Router, inklusive Ansprache des sicheren Schlüsselspeichers

Dieser Test wurde entsprechend zum ersten ausgeführt. Die Mappingsoftware wurde allerdings für das Mapping von cEMI-Paketen konfiguriert und es wurden zwei Telegramme im Klartext gesendet und zwei verschlüsselte. Entsprechend wurden erneut vier Alerts erwartet, bei denen sich die Quelladressen aber unterscheiden sollten.

```
Snort Log
10/27-17:05:16.323464 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.2:53 -> 30.25.0.1:53
10/27-17:05:18.797763 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.2:53 -> 30.25.0.1:53
10/27-17:05:22.693042 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/27-17:05:32.319968 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
```

Auch diese Telegramme entsprechen den Erwartungen. Das Wiresharklog befindet sich ebenfalls unter https://scm.cms.hu-berlin.de/tom.schuett/KNXtoIP/-/tree/main/Wireshark_Logs.

11.3 Lasttest unter Verwendung des testing Moduls

Um festzustellen, wie hoch der Durchsatz ist, wurde zu guter Letzt noch ein Lasttest durchgeführt. Da es nicht möglich war, entsprechend hohe Datenaufkommen in der physischen KNX-Installation zu erzeugen, wurden die Funktionen des `testing` Moduls genutzt. Da in der Arbeit von Weigt bereits festgestellt wurde, dass der Durchsatz des sicheren Schlüsselspeichers unter der Bandbreite des TP Busses liegt, wurden ausschließlich unverschlüsselte Pakete gesendet.

Um eine möglichst hohe Authentizität zu erzielen, wurde das Folgende aufgezeichnetes UDP Telegramm mit einem KNX/IP cEMI Telegramm als Payload verwendet:

```
Telegramm in hexadezimaler Darstellung
0e570e570019dd73/061005300011/2900/bcd01103000201008000
```

Das Paket besitzt eine Gesamtlänge von 26 Byte, wovon 8 zum UDP Header gehören, 6 zum KNX/IP Header, 2 zum cEMI Header und 10 zum eigentlichen L_Data Frame gehören. Das Paket wurde in einer Whileschleife ohne zusätzliche Verzögerung gesendet.

```

Snort Log
10/28-17:19:29.214792 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:29.370397 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:29.599726 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:29.853620 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:30.036099 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:30.235859 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:30.408772 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:30.553394 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:30.782138 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:30.919681 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:31.069848 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:31.215835 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:31.414816 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:31.569406 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:31.737638 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:31.945793 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:32.173361 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:32.334816 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:32.590828 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:32.810363 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:32.983353 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:33.131325 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:33.321628 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:33.518596 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:33.743682 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:33.971950 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:34.088987 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:34.239830 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:34.415771 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:34.589831 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:34.747368 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:34.913777 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:35.095033 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:35.267363 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:35.499217 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:19:35.649607 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53

```

Snort hat insgesamt 36 Nachrichten über eine Zeit von etwas über 6 Sekunden erfasst. Dies entspricht aufgerundet 6 Nachrichten pro Sekunde und einem Durchsatz von 6·26 Byte = 156 Bytes pro Sekunde, beziehungsweise 6·10 = 60 Byte pro Sekunde, wenn nur das L_Data Frame berücksichtigt wird. Da dies drastisch unter den von KNX-TP1 ermöglichten 9600 Baud liegt [Ass21b, Abschnitt 1, Abbildung 1], wurde der Test erneut durchgeführt, wobei das verbose Level der Mappingssoftware auf 0 gesetzt wurde.

Snort Log

```
10/28-17:24:07.764614 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:07.900660 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:08.040931 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:08.146228 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:08.306448 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:08.437669 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:08.615296 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:08.752013 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:08.898816 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:09.049437 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:09.233383 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:09.353924 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:09.496788 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:09.633088 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:09.780519 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:09.966806 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:10.124477 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:10.221492 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:10.353359 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:10.433881 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:10.577613 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:10.712492 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:10.848543 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:10.960487 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:11.084785 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:11.236497 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:11.348553 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:11.440273 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:11.582298 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:11.724539 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:11.918730 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:12.105633 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:12.261875 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:12.385122 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:12.548888 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:12.702582 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:12.854130 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:13.054642 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:13.209952 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:13.358365 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:13.551968 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:13.673375 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:13.861392 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:14.002435 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:14.136540 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
10/28-17:24:14.288536 [**] [1:1000001:1] "Found something" [**] [Priority: 0] {UDP} 30.25.17.3:53 -> 30.25.0.2:53
```

In diesem Durchlauf wurden 46 Telegramme in 6,5 Sekunden empfangen, was etwa 7 Telegrammen pro Sekunde entspricht. Somit entspricht der Durchsatz $7 \cdot 26 \text{ Byte} = 182 \text{ Byte pro Sekunde}$, beziehungsweise $7 \cdot 10 \text{ Byte} = 70 \text{ Bytes an L_Data Frames}$.

Es ist möglich, dass das Erstellen und senden der Telegramme einen Engpunkt bildet, welcher bei einer physischen Installation nicht existiert.

12 Ergebnis

In diesem Abschnitt soll betrachtet werden, ob die verschiedenen Komponenten erfolgreich verbunden werden konnten und ob das dadurch erhaltene IDS die Anforderungen erfüllt.

Zunächst lässt sich festhalten, dass die Übertragung von Telegrammen sowohl mit der USB-Bridge, als auch dem KNX/IP-Router erfolgreich funktioniert hat und die Telegramme verlässlich von der Mapping Software erfasst werden konnten. Auch der Wechsel zwischen den beiden Verbindungen über die config funktioniert ohne Probleme.

Die Javacard und der sichere Schlüsselspeicher konnten erfolgreich angebunden werden. Telegramme im cEMI Format konnten durch den Schlüsselspeicher entschlüsselt werden. Aktuell unterstützt der Schlüsselspeicher nur das cEMI Format, weshalb die Entschlüsselung nur bei Nutzung des KNX/IP-Routers zur Verfügung steht.

Die Mappingsoftware selbst ist in der Lage, empfangene Telegramme in IP-Pakete umzuwandeln und diese über ein Tun Interface einem IDS zugänglich zu machen. Dabei konnten zentrale Bestandteile wie Sender, Empfänger und Nutzdaten erfolgreich auf den IP-Standard abgebildet werden (siehe hierzu Kapitel 6). Es ist zu vermuten, dass ein IDS somit viele Abweichungen und Angriffe, insbesondere auf Anwendungsebene, erkennen kann. Um die tatsächliche Wirksamkeit bestimmen zu können, müsste das System ausgiebig gegen verschiedene Angriffe getestet werden. Da nicht alle Formate unterstützt werden, ist davon auszugehen, dass aktuell zusätzliche Angriffsvektoren existieren, diese sollten sich aber durch die Erweiterung des Mappers um zusätzliche Formate und Übertragungsmedien beheben lassen.

Auch konnte für Snort3 bestätigt werden, dass dieses die gemappten Pakete empfangen und Regeln darauf anwenden kann.

Beim Ausprobieren des Mappers konnten dabei keine Einschränkungen in der Funktionsweise der KNX-Installation beobachtet werden. Es muss allerdings berücksichtigt werden, dass die genutzte Installation wenigen Komponenten, geringen Datenverkehr und keine komplexen Instruktionen aufgewiesen hat. Es scheint dennoch unwahrscheinlich, dass das Weiterleiten der Telegramme die Funktionalität der Installation beeinflusst.

Wie in der Bachelorarbeit von Vincent Weigt beschrieben, stellt der Aufbau des Schlüsselspeichers und die kryptografischen Operationen der Javacard sicher, dass die Laufzeitschlüssel auch dann nicht kompromittiert werden, wenn ein Angreifer Zugriff auf den Raspberry Pi erhalten sollte. Somit ist es, selbst bei erfolgreichen Angriffen, nicht notwendig, die KNX-Installation neu aufzusetzen. Allerdings wäre es möglich, empfangene Pakete durch den Schlüsselspeicher dechiffrieren zu lassen. Dem kann mit den üblichen Methoden zur Gerätesicherheit entgegengewirkt werden.

Das Mapping von Telegrammen erfolgt, wie gewünscht, in Echtzeit. Allerdings liegt der maximale Durchsatz, sowohl des sicheren Schlüsselspeichers ([Wei24] Abschnitt 4), als auch des Mappings von unverschlüsselten Telegrammen, deutlich unter dem maximalen Durchsatz von KNX. Für temporäre Überlast wird dieses Problem durch die Queue aufgefangen. Für eine kommerzielle Implementierung sollte die Performance allerdings erhöht werden.

Zusammengefasst lässt sich sagen, dass das Mapping von KNX-Telegrammen auf IP zur Nutzung von Intrusion Detection Systemen, im Rahmen der Annahmen und mit den beschriebenen Einschränkungen, erfolgreich implementiert werden konnte.

13 Ausblick

Im Rahmen dieser Arbeit wurde ein POC für das Mapping erstellt und seine grundlegende Funktionalität geprüft. Um abschließend bewerten zu können, ob das Mapping von KNX Telegrammen geeignet ist, um Gefahren zu erkennen, müsste der Prototyp ausgiebig getestet werden. Hinzu kommt, dass es noch weitere Funktionen in KNX gibt, welche vom aktuellen Mapping nicht erfasst werden. Dieser Abschnitt soll deshalb weiterführende Schritte vorstellen, mit denen das Mapping erweitert werden kann.

13.1 Erweiterung des Mappers

1. *Durchsatz*: Eine der größten Einschränkungen der Mappingsoftware ist aktuell ihre Geschwindigkeit. Für einen praktischen Einsatz, insbesondere bei großen Installationen, sollte der Durchsatz optimiert werden. Hierfür kommen mehrere Ansätze infrage.

Aktuell wird jedes Telegramm durch das Betriebssystem geroutet. Demnach muss auch jedes Mal ein neuer Socket vom Betriebssystem bereitgestellt werden. Es scheint wahrscheinlich, dass dies ein Grund für den geringen Durchsatz ist. Dem könnte unter anderem durch die Nutzung persistenter Zugriffe auf die Netzwerkinterfaces entgegengewirkt werden. Beim Senden von Nachrichten an Tun Interfaces könnte zudem ohne Routing direkt in den Filedescriptor des Interfaces geschrieben werden. Neben einer Optimierung des Codes würde sich auch eine Migration in eine hardwarenähere Sprache anbieten.

2. *Weitere Medien und Formate*: In dieser Arbeit wurde lediglich das Mapping von Telegrammen über den Twisted Pair Bus untersucht. Für eine industrielle Anwendung sollten auch die anderen Medien wie etwa Powerline in das Mapping aufgenommen werden. Wie beim Vergleich von KNX und IP aufgeführt wurde, besitzt KNX zudem noch weitere Formate und Kontextinformationen, welche nicht in den Telegrammen selbst abgebildet werden. Es ist zu erwarten, dass die Genauigkeit des Mappings durch Integration dieser Daten verbessert werden kann. Auch sollte es möglich sein, Entschlüsselung und MAC-Verifikation für andere Telegrammformate als cEMI umzusetzen. Hierfür könnte eine Anpassung des Schlüsselspeichers erfolgen. Alternativ könnte auch die Mappingsoftware so angepasst werden, dass sie zunächst erhaltene Telegramme in das cEMI-Format umwandelt, um diese anschließend an den Schlüsselspeicher zu übergeben.

13.2 Weiterführende Tests

1. *Grundlegende Funktionstests*: Die zum Testen genutzte KNX-Installation besaß nur wenige Geräte und hat nur die Formate KNX_Secure und L_Data_Standard gesendet. Um eine korrekte Funktionalität des Mappers zu verifizieren, wäre es wünschenswert, die Anzahl der verwendeten Geräte zu erhöhen und möglichst viele verschiedene Formate in der Installation zu nutzen. Auch sollte getestet werden, ob die Telegrammerfassung auch dann vollständig funktioniert, wenn die überwachte Installation mehrere Buslinien verwendet.
2. *Simulation von Angriffen*: Um herauszufinden, ob ein IDS tatsächlich in der Lage ist, Angriffe zu erkennen, sollten diese ausgeführt werden. Dabei sind sowohl Angriffe gegen die KNX Installation, als auch die Mappingsoftware und den Schlüsselspeicher durchzuführen.
3. *Konfiguration des IDS*: Snort3 bietet eine Vielzahl von Möglichkeiten, mit denen das IDS auf einen konkreten Anwendungsfall angepasst werden kann. Da in dieser Arbeit keine Angriffe entwickelt wurden, ist nur eine rudimentäre Konfiguration von Snort genutzt worden. Sobald Angriffe gegen die genutzte KNX Installation gestartet wurden, kann getestet werden, in welchem Umfang die Personalisierung genutzt werden kann. Zudem kommt Snort3 mit einem großen Set von vorgefertigten Regeln. Es wäre interessant, ob diese auch auf gemappte Pakete unverändert angewandt werden können.
4. *Machine Learning*: Der Abgleich von Paketen gegen statische Regeln ist nur ein Anwendungsfall von IDS. Der große Nachteil besteht darin, dass dadurch nur Angriffe erkannt werden können, die bereits bekannt sind. Deshalb gibt es auch IDS, welche Machine Learning verwenden, um zu erkennen, wie sich ein Netzwerk üblicherweise verhält und bei Abweichungen Warnungen absetzen. Es sollte getestet werden, ob die Qualität des Mappings die Verwendung dieses Ansatzes ermöglicht.

14 Glossar

- IDS: Intrusion Detection System
- KNX: Busbasierter Gebäudeautomationsstandard der KNX-Association. KNX ist dabei die Kurform von Konnex [Autc].
- POC: Proof of Concept.
- KNX-Gerät: Eine Hardware und darauf betriebene Software, deren Konformität mit dem KNX-Standard durch die KNX-Association verifiziert wurde.
- KNX-Installation/Installation: Zusammenschluss und Konfiguration mehrerer KNX-Geräte zu einem Netzwerk.
- (KNX-)Telegramm: Eine Folge von Bytes, welche den Vorgaben der KNX-Spezifikation entspricht.
- ETS: Windowsbasiertes Konfigurationswerkzeug für die Einrichtung von KNX-Installationen.
- Abbildung: In dieser Arbeit eine Zuordnung von IP-Paketen zu KNX-Telegrammen.
- API: Application Interface, eine Schnittstelle, um auf die Funktionen einer Anwendung zuzugreifen.
- APDU: Die APDU besteht aus dem Application Layer Service Code, sowie den Nutzdaten der Anwendung [**knx'03'03'07**]
- APCI: Application Layer Service Code
- Poll Gruppe: Eine Menge von Poll Slaves [Ass21b, Abschnitt 2.2.6.1]
- Line: Eine Bezeichnung für den TP1-Bus. Es können bis zu 256 Geräte auf einer Linie zusammengeschaltet werden [Ass13, Abschnitt 3.1].
- Main-Line: Ein Bus, der mehrere Lines miteinander verbindet [Ass13, Abschnitt 3.1].
- Area: Ein Zusammenschluss mehrerer Lines durch eine Mainline [Ass13, Abschnitt 3.1].
- Domain: Ein Zusammenschluss von 15 Areas über eine Backboneline [Ass13, Abschnitt 3.1].

Literatur

- [Ass13] KNX Association. *Architecture*. Version 03.00.02. 2013.
- [Gor17] Walter Goralski. „Chapter 7 - IPv4 and IPv6 Headers“. In: *The Illustrated Network (Second Edition)*. Hrsg. von Walter Goralski. Second Edition. Boston: Morgan Kaufmann, 2017, S. 197–219. ISBN: 978-0-12-811027-0. DOI: <https://doi.org/10.1016/B978-0-12-811027-0.00007-2>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128110270000072>.
- [Feh20] Alexander Fehr. „Sicherer Schlüsselspeicher für KNX Data Secure“. Diplomarbeit. Humboldt Universität zu Berlin, 2020. URL: https://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2020-05/SARPR-2020-05_.pdf (besucht am 16.02.2025).
- [Ass21a] KNX Association. *Communication Media KNX IP*. Version 01.01.02. 2021.
- [Ass21b] KNX Association. *Communication Media Twisted Pair 1*. Version 01.03.03. 2021.
- [Ass21c] KNX Association. *Communication Medium KNX IP*. Version 01.01.02. 2021.
- [Ass21d] KNX Association. *External Message Interface*. Version 01.04.02. 2021.
- [Wei24] Vincent Weigt. „Sicherer KNX-Schlüsselspeicher auf einer Javacard“. Bachelorarbeit. Humboldtuniversität zu Berlin, 2024. URL: https://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2024-01/SAR%20PR-2024-01_.pdf (besucht am 16.10.2025).
- [Assa] KNX Association. URL: <https://www.knx.org/knx-it/per-i-professionisti/knx-benefits/knx-secure/> (besucht am 18.10.2025).
- [Assb] KNX Association. URL: <https://www.knx.org/knx-en/newsroom/en/news/Security-configuring-KNX-Secure-systems-in-ETS/> (besucht am 18.10.2025).
- [Assc] KNX Association. URL: <https://www.knx.org/knx-en/for-your-home/how-to-start/ets-home-edition/> (besucht am 25.10.2025).
- [Auta] Wikipedia Autoren. URL: <https://en.wikipedia.org/wiki/KNX> (besucht am 18.10.2025).
- [Autb] Wikipedia Autoren. URL: https://en.wikipedia.org/wiki/Explicit_Congestion_Notification (besucht am 23.10.2025).
- [Autc] Wikipedia Autoren. URL: https://de.wikipedia.org/wiki/KNX_Association (besucht am 30.10.2025).
- [Co] REINER Kartengeräte GmbH und Co. KG. URL: <https://help.reiner-sct.com/de/support/solutions/articles/101000480008> (besucht am 24.10.2025).
- [Com] The Scapy Community. URL: <https://scapy.readthedocs.io/en/latest/> (besucht am 27.10.2025).

- [Deb] SPI Inc. Debian. URL: <https://packages.debian.org/sid/libpcsclite1> (besucht am 24.10.2025).
- [Dör] Alexander Dörsam. URL: <https://www.elektro.net/41290/angriffe-auf-gebaeudeleitsysteme/>.
- [Fou] Raspberri Pi Foundation. URL: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/> (besucht am 18.10.2025).
- [Gmb] Antago GmbH. URL: <https://antago.de/knx-security/>.
- [Mik] Erik Mikac. URL: <https://www.cbtnuggets.com/blog/technology/networking/what-is-differentiated-services-code-point-dscp> (besucht am 23.10.2025).
- [Scha] Tom Schütt. URL: <https://scm.cms.hu-berlin.de/tom.schuett/KNXtoIP> (besucht am 30.10.2025).
- [Schb] Tom Schütt. *Komponentendiagramm KNX Mapper, physisch*. Selbsterstellt.
- [Schc] Tom Schütt. *Komponentendiagramm KNX Mapper, software*. Selbsterstellt.
- [Sec] Limes Security. URL: <https://limessecurity.com/de/knxlock/>.
- [Tea] Cisco Talos Detection Response Team. URL: <https://docs.snort.org/rules/options/payload/> (besucht am 24.10.2025).
- [Wei] Vincent Weigt. URL: <https://gitlab.informatik.hu-berlin.de/weigtvin/keystore> (besucht am 18.10.2025).

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 15. Februar 2026

.....