Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Security Analysis of the KNXnet/IP Secure Protocol

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

eingereicht von:	Robert Gützkow
geboren am:	[redacted]
geboren in:	[redacted]
Gutachter/innen:	Prof. Dr. Jens-Peter Redlich
	Prof. Dr. Björn Scheuermann

eingereicht am: .... verteidigt am: ....

Abstract

KNX is a standard for building automation systems that supports communication over IP. The KNXnet/IP protocol does not encrypt or authenticate the communication between devices. Connecting the KNX installation to an IP-network thus poses a high risk. The KNXnet/IP Secure protocol is supposed to remedy these security shortcomings. In 2014 Judmayer et al. [1] published a security analysis of an early and incomplete draft of the KNXnet/IP Secure protocol specification in which they identified several design flaws. In 2019 the finalized protocol specification of KNXnet/IP Secure was published as international standard ISO 22510:2019. It has to be investigated whether the past security issues have been adequately addressed and if it is a secure cryptographic protocol. This work analyzes the standard for design flaws that impact the security, including a formal analysis in eCK-PFS and model checking of state machines. The configuration software ETS5 is tested for safe defaults and secure storage of cryptographic secrets. Furthermore, black-box tests such as protocol state fuzzing and boundary-value analysis are conducted to check whether certified devices and the ETS5 conform with the specification. A risk analysis for common use cases of KNX is performed. Both the unicast and multicast KNXnet/IP Secure protocols have several conceptual security flaws. Additionally, the ETS5 stores cryptographic secrets improperly by obfuscating them with a hard-coded password (CVE-2021-36799). One tested KNX IP Secure router had a denial-of-service vulnerability (CVE-2021-37740). The behavior of the KNX devices and the ETS5 also deviate from the specification in parts that are not critical to security. Possible threats against KNX installations are identified and improvement suggestions for the protocol specification, implementation and network security are made. KNXnet/IP Secure is not a secure cryptographic protocol.

Contents

List of Figures

List of Tables

List of Acronyms

AEAD	authenticated encryption with associated data . . . . x, 8, 26, 63, 67, 73, 81, 86, 98
AKE	authenticated key exchange . . . . vii, x, 9, 10, 23, 29, 32, 40, 42, 43, 46, 49, 50, 63, 67, 68, 69, 70, 71, 72, 73, 74, 75, 78, 79, 80, 81, 82, 86, 87, 90, 92, 107
ART	asynchronous ratcheting trees 9
ASIC	application-specific integrated circuit 35, 68
BACS	building automation and control systems 1, 6
BDD	binary decision diagrams 11, 57, 58, 87
BMC	bounded model checking 11, 57, 58, 87
BSI	Bundesamt für Sicherheit in der Informationstechnik 5, 8, 12, 60, 67
cEMI	common external message interface 19, 20, 21, 22, 23, 24
CGKA	continuous group key agreement 10, 90
CIA	confidentiality, integrity and authentication 61
CRI	connection request information 101, 149, 150
CSPRNG	cryptographically secure pseudorandom number generator . . . 91
CTL	computation tree logic 11, 58
CVD	coordinated vulnerability disclosure 94, 95, 98, 100
DoS	denial of service 68, 69, 70, 100, 101, 107, 110, 146
ECC	elliptic curve cryptography 7, 8, 9, 29, 31
ECCDHP	elliptic curve computational Diffie-Hellman problem 30, 31
ECDDHP	elliptic curve decision Diffie-Hellman problem 30, 31
ECDH	elliptic curve Diffie-Hellman . vii, xiv, 29, 31, 32, 33, 68, 73, 74, 75

ECDLP	elliptic curve discrete logarithm problem	30, 31, 32, 70, 73, 74, 75, 81
eCK	extended Canetti-Krawczyk	10
EIB	European Installation Bus	1
ETS	engineering tool software	i, vii, viii, 2, 3, 4, 5, 10, 11, 16, 18, 19, 20, 22, 23, 24, 25, 39, 42, 51, 72, 81, 85, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 107, 108, 109, 110, 143, 144, 151
FDSK	factory default setup key	42, 70, 81, 91, 93, 109
FSM	finite state machine	vii, viii, 49, 54, 55, 57, 58, 59, 60, 63, 78, 79, 80, 82, 87, 88, 89, 99, 102, 103, 104, 105, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 141, 142, 145, 147
GKE	group key exchange	9, 10, 90
HBES	home and building electronic systems	1
HPAI	host protocol address information	17, 18, 22, 25, 46, 68, 81, 102, 141, 148
HVAC	heating, ventilation and air conditioning	1
IA	individual address	vii, 12, 13, 14, 15, 18, 19, 20, 21, 42, 50, 101, 149, 150
IDS	intrusion detection system	109
IoT	internet of things	60
IP	internet protocol	1, 6, 14, 15, 19, 21, 25
ISMS	information security management system	12
ISO	International Organization for Standardization	81
IV	initialization vector	vii, 94
KCI	key-compromise impersonation	10
LTL	linear temporal logic	11, 58
MAC	message authentication code	xii, 47, 48, 49, 67, 70, 71, 72, 74, 103, 124, 125, 138, 139, 144
MFA	multi-factor authentication	109
MitM	man-in-the-middle	9, 32, 68
NAC	network access control	107, 108, 109
NIST	National Institute of Standards and Technology	8, 12
PFS	perfect forward secrecy	35, 63, 74
PID	property identifier	42
PKI	public key infrastructure	81, 90
PL	powerline	14, 19
PPT	probabilistic polynomial-time	x, 37, 38, 39, 77, 124, 125

PRF	pseudo-random function	xv, 33, 34, 42, 66
RCE	remote code execution	107
RF	radio frequency	14, 19
SIEM	security information and event management	109
STS	station-to-station	9, 70, 80, 81
SUT	system under test	59, 60, 102
TP	twisted pair	1, 2, 7, 12, 14, 15, 19, 21, 24
UX	user experience	4

List of Terms

CCM	AEAD cipher mode introduced in [2] and standardized in [3]	xi, xii, xiv, 8, 26, 27, 39, 40, 44, 45, 46, 47, 48, 63, 64, 65, 66, 67, 69, 71, 72, 73, 75, 77, 81, 82, 86, 124
eCK-PFS	Formal security model for the analysis of AKE protocols introduced in [4]	i, x, xi, xii, xiii, 5, 10, 12, 26, 36, 63, 75, 76, 77, 78, 82, 124, 125, 126

List of Symbols

$\mathbb{N}$	The set of natural numbers without zero. Also denoted as $\mathbb{Z}_{\geq 1}$ in definition 3.34.	x, 37
$\mathbb{R}$	The set of real numbers.	39
λ	Security parameter $\lambda \in \mathbb{N}$ is used in the application of eCK-PFS to define the negligible function and the advantage of a PPT adversary. See definition 3.33 for details. It is equivalent to k specified in [4].	xii, 38, 39, 76, 77, 78, 125, 126
$\mathcal{A}$	Adversary used in the application of eCK-PFS to refer to an attacker modelled as PPT Turing machine. See definition 3.29 for details. It is equivalent to E specified in [4].	xi, xii, 37, 38, 39, 76, 77, 124, 125, 126
$\mathcal{P}$	Finite set of n parties' identities $\mathcal{P} = \{\hat{P}_1, \hat{P}_2, ..., \hat{P}_n\}$ in eCK-PFS [4]. See definition 3.28 for details.	xi, 37, 75, 76, 77, 124, 125
$\mathcal{I}$	Initiator $\mathcal{I}$ is a value for the session role s_{role} in eCK-PFS [4]. It indicates that the party acts as client and thus initiates the session. See definition 3.28 for details.	xi, xiii, 37, 77, 124, 125

$\mathcal{R}$	Responder $\mathcal{R}$ is a value for the session role s_{role} in eCK-PFS [4]. It indicates that the party acts as server and thus responds to requests. See definition 3.28 for details. . . . . xi, xiii, 37
s	Session s at party $\hat{P}$ is defined as tuple $(\hat{P}, s) \in \mathcal{P} \times \mathbb{N}$ and associated with a variable T_s in eCK-PFS [4]. Note that a session in eCK-PFS is the view of one party on the communication with another party. See definition 3.28 for details. . . . . xi, xiii, 37, 38, 76, 77, 124, 125, 126
T_s	State $T_s = (s_{actor}, s_{peer}, s_{role}, s_{sent}, s_{recv})$ associated with a session s in eCK-PFS [4]. See definition 3.28 for details. xi
s_{actor}	Identity of the actor $s_{actor} \in \mathcal{P}$ of a session s in eCK-PFS [4]. The session is defined from the perspective of the actor with regard to messages being sent and received, the intended communication partner, and role. See definition 3.28 for details. . . . . xi, 37, 38, 77, 125
s_{peer}	Identity of the intended peer $s_{peer} \in \mathcal{P}$ of a session s in eCK-PFS [4]. Note that the actual identity of a peer in s sending messages to s_{actor} does not have to be s_{peer}, it can also be an adversary $\mathcal{A}$. See definition 3.28 for details. . . . . xi, xiii, 37, 38, 77, 124, 125, 126
s_{role}	Role $s_{role} \in \{\mathcal{I}, \mathcal{R}\}$ the session s is executing in eCK-PFS [4]. It determines whether the s_{actor} initiates the communication with s_{peer} or waits for a request. See definition 3.28 for details. . . . x, xi, xiii, 37, 77, 124, 125
s_{sent}	Concatenation of the chronologically ordered messages $s_{sent} \in \{0,1\}^*$ sent by s_{actor} in the session s in eCK-PFS [4]. See definition 3.28 for details. . . . . xi, 37
s_{recv}	Concatenation of the chronologically ordered messages $s_{recv} \in \{0,1\}^*$ received by the s_{actor} in a session s in eCK-PFS [4]. See definition 3.28 for details. . . . . xi, 37
k	Secret symmetric key. xi, xiii, xiv, 27, 28, 46, 47, 48, 53, 69, 76, 77, 124
k_s	Session key k_s of a session s, used in the application of eCK-PFS. See "3.3.2.4 Authenticated Key Exchange" and "4.1.3 Formal Analysis with eCK-PFS" for details. . . xiii, 46, 47, 48, 49, 69, 72, 77, 124, 125, 126
$epk_{\hat{P}}$	Ephemeral public key $epk_{\hat{P}}$ of a party $\hat{P}$ in the context of a session, used in the application of eCK-PFS. See definitions 4.1 and 4.2 for details. 76, 77, 124, 125
$esk_{\hat{P}}$	Ephemeral private key $esk_{\hat{P}}$ of a party $\hat{P}$ in the context of a session, used in the application of eCK-PFS. See "3.3.2.4 Authenticated Key Exchange" and definitions 4.1 and 4.2 for details. xiii, 46, 47, 69, 76, 77, 124, 125
N	Nonce N used in CCM [3]. It should not be reused under a given key k. See definition 3.1 for details. . . . . xiv, 27, 28, 29, 40, 41, 47, 48, 49, 53, 54, 64, 65, 66, 82

P	Plaintext P used in CCM [3]. Its content will be encrypted and authenticated. See definition 3.1 for details. . . . . xiv, 27, 28, 29, 40, 41, 44, 45, 47, 48, 49, 52, 53, 64, 65, 66
A	Associated data A used in CCM [3]. Its content will be authenticated. See definition 3.1 for details. . . . . xiv, 27, 28, 29, 40, 41, 44, 45, 47, 48, 49, 52, 53, 64, 65
B_i	The i-th block B_i in CCM [3]. Its content is determined by a block formatting function β. See definition 3.1 for details. 27, 28, 29, 40, 41, 44, 45, 52, 53, 64, 65, 66, 82
Ctr_i	The i-th counter block Ctr_i in CCM [3]. Its content is determined by a counter formatting function π. See definition 3.1 for details. 27, 28, 40, 44, 45, 52, 53, 64, 65, 82, 83
T	Tag T in CCM [3]. It is the MAC for the authentication. See definition 3.1 for details. 27, 28, 65, 66
T_{len}	Tag length T_{len} as bits in CCM [3]. See definition 3.1 for details. . . . . xiv, 27, 28, 46, 47, 48, 49, 53, 65, 66
S	Keystream S used in CCM [3]. See definition 3.1 for details. . 27, 65
C	Ciphertext C used in CCM [3]. It is the encrypted and authenticated result of CCM. See definition 3.1 for details. . xiv, 27, 28, 47, 48, 49, 65, 66

List of Operators and Functions

$\text{Adv}_{\mathcal{A}}^{\Pi}(\lambda)$	Advantage of an adversary $\mathcal{A}$ to win adversary game in the eCK-PFS model for protocol Π with regard to security parameter λ [4]. The parameter $\mathcal{A}$ and Π are variables that can be replaced with the specific adversary and protocol names. See definition 3.33 for details. 38, 39, 76, 77, 78, 125, 126
$\text{negl}(\lambda)$	Negligible function (see [5, p.28]) with regard to the security parameter λ, for the eCK-PFS model. See definition 3.35 and [5, p. 28] for details. . . 39, 77, 78, 125, 126
$\text{send}(s, m)$	Query in the eCK-PFS model lets an adversary send a message m to sessions s and receive the response [4]. See definition 3.29 for details. It is equivalent to $send(s, v)$ specified in [4] except for the variable name of the message. 37, 38, 77, 124, 125

send$(s, \hat{P})$	Query in the eCK-PFS model lets an adversary activate a session s with peer $s_{peer} = \hat{P}$ in $s_{role} = \mathcal{I}$ by sending a message m [4]. See definition 3.29 for details. It is equivalent to $send(s, \hat{Q})$ specified in [4] except for the variable name of the party. 37
send$(s, \hat{P}, m)$	Query in the eCK-PFS model lets an adversary activate a session s on behalf of $s_{peer} = \hat{P}$ in $s_{role} = \mathcal{R}$ by sending a message m [4]. See definition 3.29 for details. It is equivalent to $send(s, \hat{Q}, m)$ specified in [4] except for the variable name of the party. 37, 77, 125
corrupt$(\hat{P})$	Query in the eCK-PFS model lets an adversary reveal the long-term secret key k of a party $\hat{P}$. See definition 3.29 and [4] for details. The definition is equivalent to [4] except for the variable name of the long-term secret key. 37, 38, 76, 77, 124, 125, 126
ephemeral-key(s)	Query in the eCK-PFS model to let an adversary reveal the ephemeral private keys $\{esk_{\hat{P}_1}, esk_{\hat{P}_2}\}$ of a session s. See definition 3.29 and [4] for details. The definition is equivalent to [4] except for the variable name of the ephemeral private keys. . . . 37, 38, 76, 77, 125, 126
session-key(s)	Query in the eCK-PFS model to let an adversary reveal the session key k_s of a completed session s. See definition 3.29 and [4] for details. The definition is equivalent to [4] except for the variable name of the session key. 37, 38, 77, 124, 125
test-session(s)	Query in the eCK-PFS model that randomly returns either established session key of session s or a random key that follows the probability distribution of keys generated [4]. See definition 3.29 for details. 37, 38, 77, 125, 126
$x \leftarrow y$	Assign variable x the value of y, which can be another variable, the result of a function or an expression. 27
$X \to Y$	Mapping from a set X to a set Y. 39
$x \oplus y$	Exclusive or operator that is applied bitwise on the operands. 27
$x \| y$	Concatenation of the bits in x and y. 27
$\lceil x \rceil$	Round x to the nearest integer greater than or equal to x. 28, 41
len(x)	Length of x in bits 27, 28, 34, 41, 65

β	Formatting function that returns $B_0, B_1, ..., B_r$ for use in CCM. The required parameters are not listed, because the function is not standardized. See definition 3.1 and [3] for details. xii, xiv, 26, 27, 28, 40, 41, 47, 48, 49, 53, 64, 65
π	Counter generation function that returns $Ctr_0, Ctr_1, ..., Ctr_m$ for use in CCM. The required parameters are not listed, because the function is not standardized. See definition 3.1 and [3] for details. xii, xiv, 26, 27, 28, 40, 41, 47, 48, 49, 53, 64, 65
$\text{MSB}_x(y)$	Most significant x number of bits from y 27, 28, 40, 46, 47, 64, 65, 66, 69, 77, 82, 124, 125
$\text{LSB}_x(y)$	Least significant x number of bits from y 27, 28, 64, 65, 82
$\text{PAD}_x(y)$	Pad y with zero bits until the length of y in bytes is a multiple of x. 34, 40, 41, 64, 65
$\text{INT}(x)$	The x encoded as big endian 32 bit integer. 34
$\text{Ciph}(k, I)$	Block cipher *Ciph* applied to an input I under the key k 27, 28
$\text{CCM-ENC}(k,\pi,\beta,T_{len},N,P,A)$	CCM cipher mode encryption with AES-128 as block cipher as specified in [6]. Note that KNXnet/IP Secure uses different block and counter formatting functions than the ones in [3]. See figure 5 for details. 28, 47, 48, 49, 53
$\text{CCM-DEC}(k,\pi,\beta,T_{len},N,C,A)$	CCM cipher mode decryption with AES-128 as block cipher as specified in [6]. Note that KNXnet/IP Secure uses different block and counter formatting functions than the ones in [3]. See figure 6 for details. 28, 47, 48, 49
$\text{X25519}(n, q)$	X25519 for ECDH as specified in [7], where n is a 256 bit variable containing the private key and q is a 256 bit variable containing the x-coordinate of a point on the curve. . . . 32, 33, 46, 67, 68, 69, 75, 76, 77, 124, 125

PBKDF2(p,s,c,dk_{len})	Key derivation function PBKDF2 as specified in [8]. The password p is used to derive a cryptographic key. The salt s hinders dictionary attacks, when chosen appropriately. The iteration count c determines the number of times the PRF is applied, with higher iterations increasing the computational costs for an attacker. dk_{len} is the desired length of the derived key. KNXnet/IP Secure uses HMAC-SHA-256 as the PRF [6]. 33, 34, 42, 67, 68, 70, 71, 80, 81
SHA-256(x)	SHA-256 hash function as specified in [9]. 34, 46, 47, 69, 77, 92, 124, 125

1. Introduction

KNX is a standard for home and building electronic systems (HBES) and building automation and control systems (BACS) that is developed by the KNX Association. It specifies the transmission media, hardware requirements and protocols for KNX devices, like sensors and actuators. KNX installations can be used for the automation, control and monitoring of applications such as heating, ventilation and air conditioning (HVAC), lighting, shutters, energy metering, alarm systems and access control [10], [11]. KNX is recognized as a European standard by EN 50090, EN 13321-1 and EN 13321-2 [11, p. 27], [12]. Internationally, it is standardized in ISO/IEC 14543-3-1 to 14543-3-7 and ISO 22510 [11, p. 27], [12], [6]. The KNX Association has also self-published a freely accessible version of the standard in the past [13]. However, the most recent version is v2.1 from 2013, which is missing more recent changes and extensions specified in international standards.

KNX originates from the European Installation Bus (EIB)[10]. Hence, the main transmission medium for KNX installations is twisted pair (TP). The benefit of the TP-bus is that connected devices can not only communicate over it, but be supplied with power as well. Even so, physical constraints for TP restrict the distance between devices, thus limiting the extent of KNX installations [14, p. 6], [10]. Additionally, the data rate is lower compared to other media such as Ethernet [10].

In 2008 the KNX Association extended its standard to support communication over the internet protocol (IP) with KNXnet/IP [15]. This allows the use of Ethernet (IEEE 802.3) and WLAN (IEEE 802.11) as transmission media in conjunction with TP. KNX installations can also be connected to existing IP-networks, offering the ability to perform remote configuration and management. However, this also increases the attack surface compared to the previously air gapped architecture. Unlike with KNX installation that only use TP, where an attacker would require physical access to either the devices or the bus, KNXnet/IP devices could be exposed to remote attacks, if no further security measurements are taken. Additionally, the KNXnet/IP protocol does not provide any confidentiality or authentication [16], [17]. This means when an adversary is able to gain access to the network, they could eavesdrop on the communication, impersonate devices and inject commands to take over control in the installation [17], [18]. Configuration of the devices is only protected by a password that is sent as plaintext [16], [17]. Since use cases for KNX include security applications such as alarm systems or access control, this poses a significant risk.

In order to address the lack of confidentiality and authentication the KNXnet/IP Secure protocol was developed. Judmayer et al. [1] discovered several security flaws in their analysis of the early draft proposal "Application Note 159/13 v02" in 2014. The KNX standard v2.1 from 2013 contains a more recent version of the draft, "Application Note 159/13 v04" [19]. It has largely the same security issues. Newer drafts are not publicly available. The finalized protocol design was submitted to the International Organization for Standardization in 2017 [20]. ISO 22510, published in 2019, is the approved international standard that contains the specification of KNXnet/IP and KNXnet/IP Secure [6]. KNX IP Secure devices have been commercially available since [21, pp. 44].

KNXnet/IP Secure only protects communication over IP, hence a different solution is necessary for TP and other media. The KNX Association developed KNXnet/Data Secure for this purpose. It is meant to provide authentication, confidentiality and access control [22, p. 9]. The combination of KNXnet/IP Secure and KNXnet/Data Secure is referred to as KNX Secure [23]. According to the KNX Association "KNX Secure is the only security standard for smart homes and buildings that meets the world's highest cyber security requirements." [21, p. 5].

1.1. Problem Statement

According to ISO 22510:2019, KNXnet/IP Secure has the objective to ensure confidentiality, authentication, data integrity and freshness [6]. In particular unicast communication is supposed to provide mutual authentication and multicast communication the authentication of group membership [6]. Research by Judmayer et al. [1] has shown security issues in an early draft of the protocol specification, that affected both the confidentiality and authentication. This raises the question whether ISO 22510:2019 specifies a secure cryptographic protocol. It appears that no security analysis of the international standard has been published at the time of writing.

Devices with support for KNXnet/IP Secure are being marketed for their supposed security properties [11]. The KNX Association showcased projects that use KNXnet/IP Secure to protect the communication of applications such as intrusion alarm systems and smoke detectors [11]. Uses case like these highlight that a failure to ensure confidential and authenticated communication could pose a risk to both lives and property. The KNX Association requires manufacturers to submit devices for a certification process in order for them to use the KNX trademark [24]. However, it is not publicly documented what requirements devices have to meet with regard to KNXnet/IP Secure. Hence, it is unclear whether the software is thoroughly tested for conformance with the protocol specification and if there are any measurements taken to identify vulnerabilities in the implementation.

The KNX devices are configured and managed with the engineering tool software (ETS) software developed by the KNX Association. Due to the protocol design of KNXnet/IP Secure, the ETS has to store cryptographic secrets used by the KNX devices. It is necessary that the ETS does so securely as it is a lucrative target for attackers trying to gain control over KNX installations. There does not appear to be a current audit or research published on the security of the ETS.

In summary, the general problem is that the finalized specification of KNXnet/IP Secure and its implementations have not received substantial attention by security researchers. This work seeks to address the problem by providing a security analysis of the KNXnet/IP Secure protocol and its applications. The purpose is to identify issues that can be solved in future version of the protocol and its software implementations, improving the security of KNX installations.

1.2. Research Questions

The research questions this work intends to answer are the following:

1. Are the unicast and multicast protocol of KNXnet/IP Secure, specified in ISO 22510:2019, considered secure according to the current state of the art in cryptography?
2. If design flaws with a security impact exist in the specification of KNXnet/IP Secure, can they be practically exploited or are they only of theoretical nature?
3. Does the ETS5 (v5.7.x) have design flaws that impact the security, in particular with regard to storage of cryptographic secrets, defaults for the configuration of KNX devices and software updates?
4. Is the ETS5 (v5.7.x) conforming with the KNXnet/IP Secure specification?
5. Are certified KNX devices with KNXnet/IP Secure support conforming with the specification?
6. If non-conformance with the standard occurs in either the ETS or KNX devices, does this impact the security?
7. What risks are KNX installations exposed to and do recommendations by the KNX Association for securing KNX installations mitigate them?

KNXnet/Data Secure is not being analyzed in this work. ETS6 (v6.0.0) was released when the research was concluded, hence no in-depth analysis was conducted for this version. Side-channel attacks on implementations of KNXnet/IP Secure are out of scope.

1.3. Research Methodology

The research questions are centered around the protocol specification, ETS, certified KNX devices and risks KNX installations are exposed to. Each of these four topics requires a different research methodology to answer the associated questions. An overview is provided in this section, while a detailed explanation is given in the respective chapters.

For the analysis of the KNXnet/IP Secure protocol, it is checked whether the cryptographic primitives are used correctly. This is accomplished by verifying if the assumptions hold, under which the primitives have been proven to be secure. The claimed security properties of the protocol can then be evaluated. Additionally, shortcomings in the writing of the standard can be identified, which includes ambiguities and undefined aspects. It is problematic if the specification leaves room for interpretation in parts that have an impact on the security, because a conforming implementation could have worse security properties than intended. Furthermore, model checking is applied to the state machines to formally verify if the textual description matches the specified model and that security relevant states, such as the authentication of a peer, are only reached when they are supposed to. Finally, a formal cryptographic model is used to evaluate if an adversary with far-reaching capabilities would have a non-negligible chance of compromising a session. Based on the results it can be concluded whether ISO 22510:2019 specifies a secure cryptographic protocol from both an applied and theoretic standpoint.

However, even when the specification is free of security issues, implementations of KNXnet/IP Secure could be insecure. This refers to both the implementation of the protocol, but also the surrounding systems it relies on. Hence, the second topic is the analysis of the ETS. It serves as both an exemplary implementation of a client in the KNXnet/IP Secure protocol and it is responsible for managing the KNX devices, which is why it stores cryptographic secrets related to them. In order to ensure that the implementation matches the specification, a series of black-box tests with boundary-value analysis are performed. The test harness is created from the specified requirements, frame formats and state machines in the ISO 22510:2019 standard. A black-box test is chosen over a white- or gray-box approach, because the same implementation can be reused to test the KNX devices. Those do not have binaries or source code are available, hence a black-box test is necessary. Behavior that deviates from the standard can then be analyzed for its impact on the security. Since the ETS takes on a special role as the device management software, responsible for generating and safely storing cryptographic secrets used by KNXnet/IP Secure, the analysis has to extend beyond the protocol implementation. Thus, the ETS is reverse engineered, decompiled and deobfuscated to investigate whether the cryptographic secrets are stored securely and if the update mechanism by the ETS ensures that an adversary cannot trick the software into installing a malicious update. Moreover, it is checked if the software uses defaults that ensure a secure configuration of the KNX devices or if there is a potential for misconfiguration due to the provided user experience (UX). The results allow to identify whether the implementation of the ETS conforms with ISO 22510:2019 and if it has design flaws that impact the security.

The third topic is the analysis of certified KNX IP Secure routers, which take on the role of servers for unicast communication. This test is conducted with the devices SCN-IP100.03 by MDT and KNX IP Router 752 secure by Weinzierl. The choice for these two device were made based on features, availability, and budgetary limitations. Black-box testing is employed to check for conformance with the specification. The test harness is similar to the one for the ETS, except that it implements tests for the server-side. A black-box test is necessary, as no source code, open firmware or debug interfaces are available. Protocol state fuzzing is used to determine the state machines that the devices implement. The inferred finite state machines are then compared to those from the specification through model checking. Behavior that deviates from the intended handshake can be identified. Based on the results it can be determined whether the certification process ensures conformance with the standard and if the tested KNX devices have design flaws that can be practically exploited.

A risk assessment is the fourth and final topic. It is supposed to determine threats that KNX installations are exposed to. The analysis process from a national standard is applied. The uses cases are inspired by real projects the KNX Association has showcased. Based on the findings it can be evaluated whether the recommendations by the KNX Association are adequate to ensure safe operation. In particular, it can be determined if KNXnet/IP Secure fulfills its intended purpose of addressing the shortcomings of KNXnet/IP or if additional measurements are necessary because previous parts have identified issues in the protocol design or implementation.

In summary, the methodology ranges from practical software tests to formal cryptographic models. The results are reproducible and verifiable. Problems identified through this approach are of general significance for KNX and not strictly limited to specific devices.

1.4. Thesis Structure

"Related Work" shows the research this thesis builds upon and places it in the context of related publications. This includes research about the security of KNX, cryptographic primitives, formal cryptographic models, authenticated key exchange, group key exchange, model checking, black-box testing and risk assessment.

"Background" gives an overview of knowledge that the reader will likely require to understand the thesis. It explains the basic concepts of KNXnet/IP, which provide insight into the problems KNXnet/IP Secure is supposed to solve. Furthermore, cryptographic primitives used by KNXnet/IP Secure are introduced as well as the eCK-PFS model for analyzing the security of key exchange protocols. The key elements of KNXnet/IP Secure are explained to give an outline of how the unicast and multicast protocols work. Additionally, an overview of model checking with NuXMV, protocol state fuzzing and risk analysis with BSI 200-3 is given.

"Analysis of KNXnet/IP Secure in ISO 22510:2019" examines both the unicast and multicast protocol of KNXnet/IP Secure for design flaws. The security properties of the protocols are determined and ambiguities identified that could affect how manufacturers implement the protocols. Model checking with NuXMV is used to determine whether the state machines are properly specified. A formal analysis is conducted in the eCK-PFS model.

"Device Management with the ETS5" analyzes the configuration process of KNX devices with the ETS5. It is checked whether the program provides safe defaults that ensure a secure operation of KNX installations. Furthermore, the ETS5 is reverse engineered to investigate the storage of cryptographic secrets and the update mechanism. A conformance test with boundary-value analysis is performed based on the ISO 22510:2019 standard.

"Analysis of Certified Devices" performs black-box testing with protocol state fuzzing and boundary-value analysis to check for conformance with the specification. Ambiguities that have been identified in the standard are used to test edge cases. The result of the protocol state fuzzing is evaluated by applying model checking with NuXMV, in order to compare the finite state machines with the specification. Additionally, the devices are checked for behavior that could indicate the presence of vulnerabilities.

"Risk Analysis with BSI 200-3" applies the standard by the Bundesamt für Sicherheit in der Informationstechnik (BSI) to identify threats that KNX installations face and whether guidelines provided by the KNX Association are enough to ensure secure

operation. The risk assessment uses scenarios based on real projects presented in the KNX journals.
“Evaluation” verifies whether the posed research questions have been answered and summarizes the findings.
“Conclusion and Future Work” draws conclusions from the findings, in particular whether KNXnet/IP Secure is a suitable solution to address the shortcomings of KNXnet/IP. Areas that require further research are identified, including approaches that could lead to more insights that were not covered in this work.

2. Related Work

This chapter provides an overview of the literature that the thesis builds upon. Contributions made in this work are placed into context. The publications include prior research about the security of KNXnet/IP and the draft of KNXnet/IP Secure as well as methods used to analyze the standard and its applications in later chapters. Hence, papers from a wide range of research areas are included, from cryptographic primitives to risk assessment.

2.1. KNX

In 2006 Granzer et al. [16] published their research about security shortcomings in BACS protocols, including LonWorks, BACnet and KNX/EIB. They proposed the EIBSec protocol to address the issues in KNX/EIB [16]. Their research demonstrates the long-existing problem of insecure communication in BACS and KNX in particular. It is an important publication that has sparked research into the security of KNX.

The KNX standard was extended to include support for communication over IP in 2008 [15]. BACnet and LonWorks provide protocols based on IP as well [25]. Hence, Granzer et al. [25] analyzed existing solutions and provided a generic security concept that could be transparently applied to different BACS protocols to improve their security, including KNXnet/IP. They note that KNXnet/IP does not provide effective protection, but rather relies on guidelines for isolating the installation and security by obscurity [25]. This publication from 2009 tries to solve the same problems as KNXnet/IP Secure specified in ISO 22510:2019. Therefore, it lends itself for a comparison. Security by obscurity is also a design issue that will be revisited in this work.

The lack of security in KNXnet/IP has been demonstrated with practical attacks by Antonini et al. [17] and Molina [18] in 2014. Both have shown that an attacker with access to the network can potentially control the KNX installation, if no additional security measurements have been taken. Antonini et al. [17] also explained in detail how the password protection for device configuration, which is the only security mechanism in KNXnet/IP, can be circumvented by observing the network traffic. That is because the password is sent unencrypted. The publications highlight the need for secure communication. Antonini et al. [17] concluded that fixing the underlying weakness of the system requires a drastic improvement of the protocol and devices.

A central paper for this thesis is the publication by Judmayer et al. [1]. They analyzed the draft proposal “Application Note 159/13 v02” from 2013, which specifies an early version of KNXnet/IP Secure. Issues in the protocol design were found that affected the confidentiality and authentication [1]. However, given that the analysis was conducted on a draft, the protocol specification was subject to change and problems that have been found might not apply to ISO 22510:2019. Since the KNX Association submitted the final protocol specification to the ISO in 2017, they had roughly 3 years to address the issues [20]. This thesis continues the research that Judmayer et al. started. A key contribution in this work is the analysis of the finalized protocol specification from ISO 22510:2019. The findings from Judmayer et al. are discussed in the “Analysis of KNXnet/IP Secure in ISO 22510:2019” chapter, for both issues that have been fixed or still exist.

There have only been a few publications since, that address security shortcomings in KNXnet/IP or KNXnet/IP Secure. While KNXnet/IP Secure was not yet finalized, Glanzer et al. [26] suggested modifications to increase the resilience against replay attacks and introduce redundancy for high availability. Seifried et al. [27] identified the issue of KNXnet/IP being only specified for IPv4 and suggested how IPv6 support could be integrated. Furthermore, they compared the KNXnet/IP Secure draft from “Application Note 159/13 v04” [19] to IPSec. A combination of the two was proposed for securing the communication [27]. The thesis of Goltz [28] evaluates risks posed to the TP-bus of KNX. While not directly related, it inspired the inclusion of a risk analysis with BSI 200-3 in this work. Similar to the research by Antonini et al. [17], Vacherot [29] demonstrates the risk of plain KNXnet/IP communication. The paper presents a fuzzer which can be used to test KNXnet/IP devices and consequentially find vulnerabilities. Identifying non-conforming behavior is one of the goals of this thesis, and thus closely related. However, a different approach was chosen as explained in the “2.11 Black-Box Tests” section and “Analysis of Certified Devices” chapter.

2.2. Cryptographic Fundamentals

Cryptography is a field that is at the core of the KNXnet/IP Secure analysis. Basic knowledge of cryptography is required to understand the thesis, as it is not within the scope of this work to explain the fundamentals. Readers are referred to the publications of Boneh and Shoup [5], Katz and Lindell [30], Menezes et al. [31] and Hankerson et al. [32] for elementary information on topics such as hashing, ciphers, key derivation, message authentication codes, security notions and elliptic curve cryptography (ECC).

2.3. Symmetric Cryptography

Bellare and Namprempre [33], and Krawczyk [34] provided insight into security notions and analyzed the properties of authenticated encryption based on generic composition of encryption and authentication. The findings are helpful to assess the security notions symmetric encryption in KNXnet/IP Secure should satisfy.

Whiting et al. [2] developed the CCM cipher mode, which is used by KNXnet/IP Secure [6]. It provides authenticated encryption with an authenticate-and-encrypt composition [2]. A security proof for it was provided by Jonsson [35]. While Rogaway [36] criticized design choices made for CCM as well as security claims made by the original authors, no "grave or urgent problems" [36] were found that affect the cipher mode. The assessment of Rogaway is relevant for the analysis of KNXnet/IP Secure as there are pitfalls to applying CCM properly with regard to the parametrization. CCM was made a standard by the National Institute of Standards and Technology (NIST) with Special Publication 800-38C [3] which requires it to be used with AES [37] as block cipher. In [38] Rogaway renewed his criticism of CCM, while acknowledging that it is a provably-secure AEAD scheme. One particular problem that the publication highlights, is that the NIST standard does not require the use of the canonical formatting and counter-generating functions [38]. Not relying on them, and using custom functions instead, can adversely affect the security properties. Hence, this is an issue that may concern KNXnet/IP Secure and is thus investigated in this work.

2.4. Asymmetric Cryptography

KNXnet/IP Secure relies on an elliptic curve for asymmetric cryptography [6]. Notable work about them has been published by Miller [39], Koblitz [40], and Koblitz, Menezes and Vanstone [41] among many others. While those publications and related work are relevant to the field, the thesis is mainly focused on Curve25519 specified by Bernstein [42]. This is the curve used by KNXnet/IP Secure. Similar to CCM, the thesis is concerned with the correct application of it. As previously stated, side-channel attacks on the implementation are out of scope. Hence, publications such as Genkin et al. [43] are not considered in this work.

2.5. Guidelines for Cryptography

National institutions like NIST and the BSI publish guidelines for the use of cryptography to ensure that reasonable security standards are met. The BSI provides recommendations for cryptographic primitives and key lengths in TR-02102 [44], as does NIST in Special Publication 800-131A [45]. Guidelines for ECC can be found in TR-03111 [46]. KNXnet/IP Secure is compared to the recommendations set forth by these documents to determine whether they are fulfilled or even exceeded.

2.6. Authenticated Key Exchange

When two parties want to ensure confidentiality of the communication between each other, they need to agree on a key that they can use to encrypt their messages. The key needs to be known only to them. Key-agreement protocols solve this problem. One of the most notable publications on asymmetric cryptography is [47] by Diffie and

Hellman. The Diffie-Hellman key exchange allows to establish a shared secret [47]. However, the plain Diffie-Hellman key exchange is susceptible to an active attacker that acts as man-in-the-middle (MitM), because it does not provide authentication. The attacker can impersonate the respective communication partner of the parties and thus establish shared secrets with both, while the legitimate parties are unaware that they are not communicating with the intended partner. Messages that were supposed to be exchanged confidentially between the parties can therefore be read by the attacker. An authenticated key exchange (AKE) tries to ensure that the involved parties can verify the identity of the communication partner [48]. One of the early AKE protocols is the station-to-station (STS) protocol by Diffie et al. [48]. This particular AKE is referenced by drafts of KNXnet/IP Secure [19][49] and thus of interest. Publications by Wilson and Menezes [50], Choo [51], Cremers [4] and Lipp et al. [52] provide insight into the security properties that can be expected from AKE protocols and need to be evaluated for KNXnet/IP Secure.

Contributions to the research on the security of AKE protocols are made with the analysis of KNXnet/IP Secure in chapter "Analysis of KNXnet/IP Secure in ISO 22510:2019".

2.7. Group Key Exchange

Secure communication does not have to be limited to two parties. If keys are not pre-shared, then multicast and broadcast communication require a way to establish a key for the group as well. Protocols that solve this problem are called group key exchange (GKE).

KNXnet/IP Secure supports multicast communication, which is supposed to ensure confidentiality and authentication of group membership [6]. In order to place the protocol design into context, it has to be compared to past and present research. This allows to identify issues and potential for improvements.

The GKE protocols not only have to perform the contributory key agreement, but also face the challenge of scalability in relation to the number of members in the group, including communication and computational complexity. Steer and Strawczynski [53] published a protocol that allows to establish a shared secret. However, the STR protocol only supports static groups and scales linearly in complexity with the number of group members [54]. GKE for dynamic groups that also push for more efficient solutions are the protocol by Burmester and Desmedt [55], CLIQUES by Steiner et al. [56], TGDH by Kim et al. [57], [58] and the extended STR by Kim et al. [54], which enhances the work by Steer and Strawczynski [53] to include support for dynamic groups and improve efficiency. Manulis [59] published optimizations for the aforementioned protocols and modified them to use ECC. Each of the protocols has trade-offs between the required number of messages and rounds to perform the GKE, computational and memory complexity, and handling of dynamic changes to the group membership [59].

More recent publications provide advances in security, such as Cohn-Gordon et al. [60]. They showed how forward secrecy and post-compromise security can be accomplished in group messaging with asynchronous ratcheting trees (ART) [60]. Similarly, Alwen et al. conducted research on continuous group key agreement (CGKA). This resulted in publications about the analysis of the TreeKEM protocol [61], CGKA as cryptographic primitive [62] and the security of the Tainted TreeKEM protocol [63].

The thesis contributes the analysis of the multicast protocol of KNXnet/IP Secure in chapter “Analysis of KNXnet/IP Secure in ISO 22510:2019”, which includes a comparison with GKE protocols as well as suggestions to improve its design based on the previously mentioned research.

2.8. Cryptographic Models

Evaluating whether cryptographic protocols are secure, requires a precise definition of what that term means. Additionally, a sensible abstraction is required that allows to perform the analysis. This can be accomplished by formulating a model that defines an adversary and their abilities as well as conditions that they must reach to break the protocol. If it can be proven that the adversary is unable to break the protocol, then it is considered secure in the given model.

Notable models for formally proving the security of protocols have been published by Dolev and Yao [64], Bellare and Rogaway [65], and Canetti and Krawczyk [66]. The Canetti-Krawczyk model has been refined by LaMacchia et al. [67] to include further attack classes, including key-compromise impersonation (KCI). This is known as the extended Canetti-Krawczyk (eCK) model. Further enhancement were developed by Cremers et al. [4] which resulted in the eCK-PFS model. It allows to prove that a protocol provides perfect forward secrecy [4]. An updated version of the same paper with minor corrections has been published in [68].

The eCK-PFS model is utilized in chapter “Analysis of KNXnet/IP Secure in ISO 22510:2019” to analyze KNXnet/IP Secure. An introduction is provided in the “Background” chapter. Swanson’s thesis [69] serves as inspiration for the formal analysis, as it applies the eCK model on other AKE protocols.

2.9. Attacks Against Weak Cryptography

KNXnet/IP Secure uses password-based authentication and derives cryptographic keys from the passwords using PBKDF2 with HMAC-SHA-256 [6]. It is of interest to evaluate if the protocol is susceptible to offline attacks, where an attacker records the communication of parties and attempts to determine what the password is. If such an attack is theoretically possible, it can indicate a design flaw, but it does not necessarily mean that it can be practically exploited. The required time, computation power and financial resources might not be attainable. The thesis seeks to answer whether there is a realistic risk of the KNXnet/IP Secure authentication being broken. Hence, the research by Visconti et al. [70], [71] and Choi et al. [72] are of interest for the optimized implementation of password cracking should an offline attack be possible.

The ETS5 provides the option to export project information, including cryptographic secrets. It applies the PKZIP stream cipher to encrypt the sensitive information. Biham and Kocher [73] discovered that the PKZIP cipher is susceptible to a known plaintext attack. The research by Stay [74] provides further improvements on the attack, in particular to address the challenge of compression being applied prior to the encryption. These publication are relevant because it is evaluated if the exported projects by the ETS5 can be decrypted with the described attacks.

2.10. Model Checking

The NuXMV software [75] is used in chapter "Analysis of KNXnet/IP Secure in ISO 22510:2019" to check properties of the state machines specified in KNXnet/IP Secure, and in chapter "Analysis of Certified Devices" to compare the inferred state machines of real devices against the specification. Although no new methods for model checking are contributed in this work, the literature that forms the basis of NuXMV is relevant for the correct application of the tool and for interpreting the results.

Temporal logics are utilized in NuXMV to express conditions for the formal verification of a given model. One is linear temporal logic (LTL) developed by Pnueli [76] and the other computation tree logic (CTL) by Clarke and Emerson [77]. Important underlying techniques for the symbolic model checking of CTLs are Bryant's binary decision diagrams (BDD) [78] and Biere et al.'s bounded model checking (BMC) [79] for LTL. The work by Clarke et al. [80] determined how the bound for BMC has to be chosen in order to ensure that a counter example can be found, if one exists. This is important, because if BMC does not produce a counter example, it does not necessarily mean that the LTL formula is satisfied. It could be that the bound is too low to find a counter example. Definitions and background information required to understand the model checking with NuXMV are provided in the "Background" chapter.

2.11. Black-Box Tests

Testing embedded devices like the KNX IP Secure routers, without using a debug interface or having knowledge about the firmware, is a challenge. All checks have to be performed over the network and responses or lack thereof need to be interpreted. The goal of the black-box testing in this thesis is primarily to check whether the KNX devices conform with the specification and if deviations from it have security implications. De Ruiter and Poll [81] developed a technique called "protocol state fuzzing" that allows to infer the state machine of a device. Their open source implementation of the StateLearner [82] based on LearnLib [83] provides a foundation for the conformance tests in this work. A protocol state fuzzer for KNXnet/IP Secure is developed in this work, which can be used to infer the state machines of KNX devices. The result can be compared with the specification, which allows to identify non-conforming behavior. Further research that applies protocol state fuzzing has been published by Fiterău-Broştean et al. [84]. Their work on identifying issues in implementations of DTLS serves as a reference for the type of problems that can be discovered with this technique.

2.12. Risk Analysis

The risk analysis in this work is based on the standard BSI 200-3 [85], [86] which specifies the methodology. It builds upon the standard BSI 200-1 [87], [88] for information security management systems (ISMSs), BSI 200-2 [89], [90], that specifies the process for creating a security concept, and the “IT-Grundschutz” compendium [91], which provides building blocks for the risks analysis and mitigation. This contribution allows to determine what risks KNX installations are exposed to and how well the guidelines provided by the KNX Association protect against them. Additionally, it helps to determine whether KNXnet/IP Secure mitigates prevalent risks or if additional measurements are required.

2.13. Network Security

The BSI publishes guidelines for securing the internal network and providing safe remote access. The former is ISi-LANA [92], the latter ISi-Fern [93]. A more recent approach to network security, called “Zero-Trust”, is presented in NIST’s Special Publication 800-207 [94] and CISA’s draft [95]. Since KNX installations that are connected to an IP-network are exposed to the similar risks as regular computers, the literature is relevant for building a secure architecture and evaluating if existing solutions are on par with the requirements. This knowledge is applied as part of the risk analysis and informs the improvement suggestions for the KNX installation guides contributed in this work.

3. Background

Knowledge about the KNX topology and internal workings of KNXnet/IP are necessary to understand KNXnet/IP Secure and its analysis. Therefore, this chapter illustrates the features of KNXnet/IP and how it integrates into KNX installations, as well as the security problems that arise. The cryptographic primitives used by KNXnet/IP Secure are introduced, followed by the security properties and eCK-PFS model for the analysis. Essential parts of the KNXnet/IP Secure specification from ISO 22510:2019 are summarized. Additionally, it is detailed how model checking with NuXMV, protocol state fuzzing and risk analysis with BSI 200-3 work.

3.1. KNX Topology and KNXnet/IP

KNX specifies a logical topology for its installations, which is depicted in figure 1. Each device is identified by a 16 bit individual address (IA) [96], [6]. It is possible for devices to have additional IAs [6, p. 52], for services that they provide. Hence, there can be at most 65536 devices in one KNX installation [96]. The logical topology describes a hierarchy consisting of lines, areas and the backbone [96]. It resembles the physical topology of installations that use TP as medium. Hence, the terminology and architecture shown in figure 1 are best explained based on it. The TP cable creates

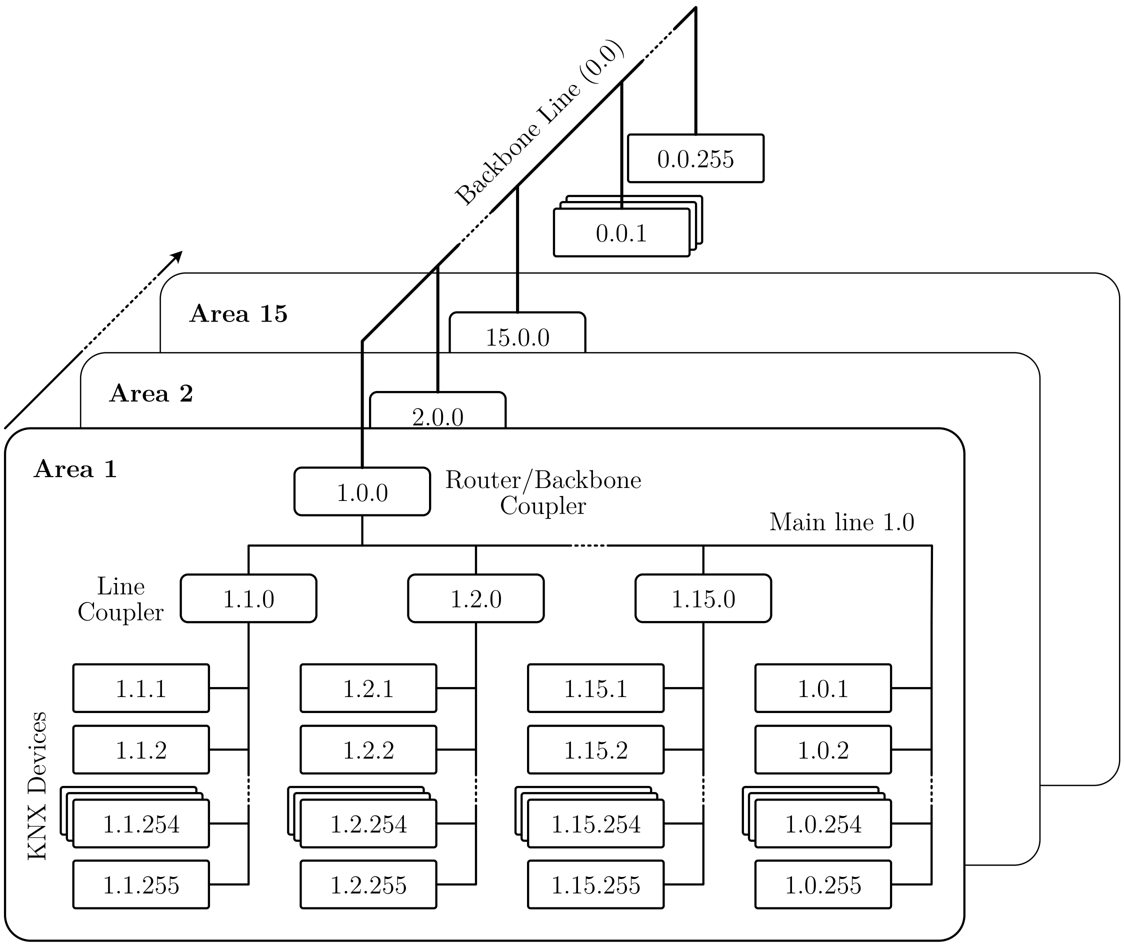


Figure 1: The KNX topology, based on [96, p. 10]

a physical segment which is referred to as a line [96]. Two lines can be connected to each other with a device called router, which routes frames from one line to another if the recipient is not located in the same line as the sender [96]. An area is formed when multiple lines are connected to an additional line. This single line is referred to as the main line of the area [96]. Routers that are used to form an area are known as line couplers [96]. Multiple areas can also be combined with routers, by connecting their main lines to the main line of one particular area [96]. Those routers are called backbone couplers [14, p. 22]. The main line of that area is referred to as the backbone line of the KNX installation [96]. Devices can be directly connected to all lines, including the backbone line and other main lines [14, p. 23], [96, p. 10].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Area Address				Line Address				Device Address							

Figure 2: Structure of IA [97, p. 6]

The logical topology is a tree, as can be seen in figure 1. This structure is defined by the IA, shown in figure 2. It is important to note that the logical topology applies regardless of the medium being used and does not have to be identical to the physical topology. All data structures of KNX are specified as big endian, such as figure 2, unless otherwise indicated. The least significant byte of the IA contains the area and line, both encoded in 4 bit respectively [97, p. 6]. Hence, there can be at most 16 areas and within each 16 lines. This part of the IA is referred to as the subnetwork address [97, p. 6]. The remaining byte allows 256 unique addresses for the devices within the subnetwork. For the textual representation of the IA, as shown in figure 1, the area, line and device address are each converted from binary to decimal and separated by dots.

There are additional rules for the IA assignment. The area that contains the backbone line has the area address set to zero [97, p. 7]. Main lines are required to have the line address set to zero [97, p. 7]. Hence, the backbone line is identified by the 0.0 subnetwork address. Routers and line couplers are required to have the device address set to zero [97, p. 6]. All other device types may not have a device address of zero [97, p. 6].

KNX specifies twisted pair (TP), powerline (PL) and radio frequency (RF) as communications media [96, p. 7], which all adhere to the logical topology. They can be combined within the same installation [96, p. 7]. KNXnet/IP is an extension of the standard that introduces communication based on the internet protocol (IP) [15]. It allows using additional media, such as Ethernet (IEEE 802.3) and WLAN (IEEE 802.11) [96, p. 7]. IP already provides a logical topology in the form of IP addresses. IPv4 and IPv6 also allow to address many more devices than the IA. KNXnet/IP thus not only specifies a protocol and the services it provides, but also the integration of IP into the logical topology of KNX. A short introduction to KNXnet/IP is given in the following paragraphs.

Since the KNXnet/IP protocol stack introduces new physical layers, additional system devices with matching interfaces are required for the use as backbone and line couplers. They are referred to as KNXnet/IP routers [6, p. 8]. Commonly these devices provide one RJ-45 interface for Ethernet and a TP interface. Unlike other communication media, such as TP, the specification does not permit using KNXnet/IP throughout the KNX installation [15, p. 4], [6, p. 81]. It has to be used in conjunction with TP, PL or RF [15, p. 4]. Hence, this poses the question of where KNXnet/IP fits into the logical topology. KNXnet/IP is meant to be used as a fast backbone [6, p. 81]. Therefore, the specification restricts where KNXnet/IP is allowed to be used. KNXnet/IP routers may not be utilized as line couplers when there is already a KNXnet/IP router in place as backbone coupler for the same area [6, p. 81], [98, p. 5]. The inverse is also not permitted. When there is already a KNXnet/IP router used as line coupler, then KNXnet/IP router may not be placed as backbone coupler in the same area [6, p. 81], [98, p. 5]. This disallows any KNXnet/IP routers to be located in the hierarchy below another KNXnet/IP router. Furthermore, if a KNXnet/IP device is assigned an IA, then all parts of the subnetwork leading to the device are required to only contain KNXnet/IP devices [98, p. 5]. In practice this means that

KNXnet/IP is used for the backbone line, down to either a KNXnet/IP router that provides an interface to another communication medium or a non-system KNXnet/IP device. Figure 3 illustrates this with an example topology, where both KNXnet/IP and TP are used.

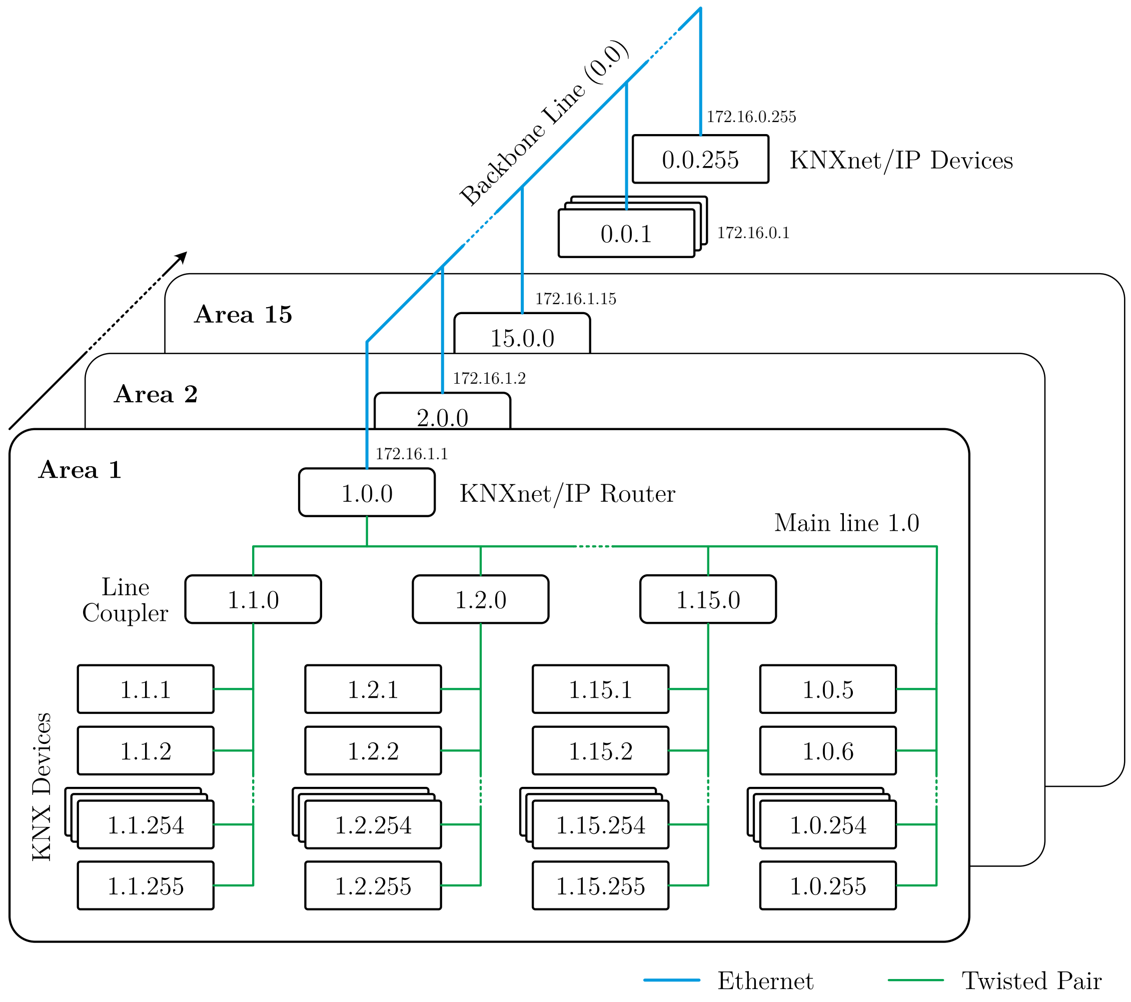


Figure 3: KNXnet/IP in KNX installation

Besides the IAs that identify the KNXnet/IP devices, they are also assigned IP addresses. Since the IAs are used for the identification of the device within the logical topology of the KNX installation, the larger space of the IP addresses does not permit to use more KNX devices. However, the KNXnet/IP devices can coexist with any number of devices on the IP-network that are not part of the KNX installation.

The KNXnet/IP protocol stack is depicted in figure 4. It is an application layer protocol, that may use both TCP and UDP in the transport layer. Supporting TCP is optional for devices that do not implement KNXnet/IP Secure [6, p. 7]. It also relies on additional protocols that are not shown in the figure, including ARP, ICMP, IGMP, and DHCP or BootP [6, p. 7], [99, p. 8]. The ISO 22510:2019 standard specifies the services that KNXnet/IP router, devices, and management tools, like the ETS, have to provide [6, p. 8]. The requirements are depicted in table 1. The following subsection summarize the features that these services provide, when KNXnet/IP Secure is not used. The purpose is to give a picture of how the KNXnet/IP devices operate, including security issues that arise. Changes to the KNXnet/IP layer that are introduced with KNXnet/IP Secure are explained in the section “3.3 KNXnet/IP Secure in ISO 22510:2019”.

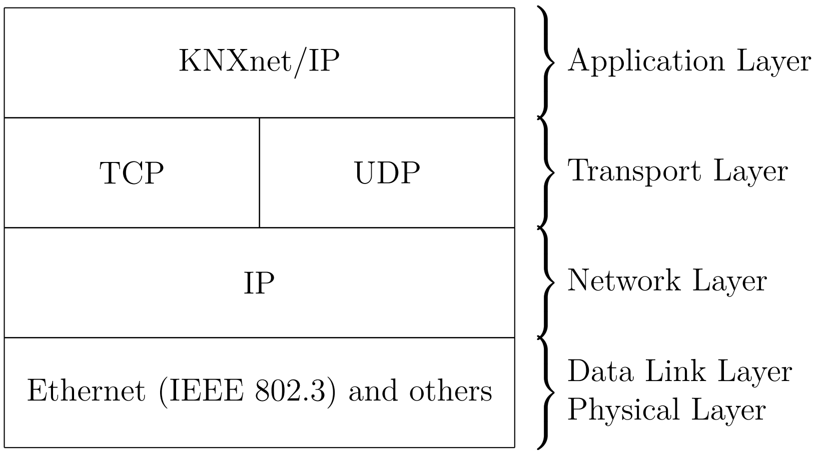


Figure 4: KNXnet/IP protocol stack [6, p. 39, p. 102]

Service Families	**Device Class**		
	Management Tools	KNXnet/IP Router	KNXnet/IP Device
Core	M	M	M
Device Management	M	M	M
Tunneling	M	M	O
Routing	M	M	O
Remote Diagnosis and Configuration	M	O*	O
Secured Communication	M	O	O

Table 1: Implementation requirements for device classes, based on [6, p. 8]. “M” stands for mandatory and “O” for optional. The entry marked with * may not be implemented when secure communication is supported.

3.1.1. Core

The Core service family defines the way KNXnet/IP devices can find each other, learn which services they offer and how to access them. The specification introduces the abstraction of endpoints for this purpose. Each device needs to have one discovery endpoint, through which it can be found by other devices and supply information about itself [6, pp. 10-12]. For every subnetwork the device is connected to, it has to provide a service container which implements at least one service related to the subnetwork [6, p. 11, p. 49]. The service container is represented by its control endpoint [6, p. 11]. It shall allow to establish and control a connection for accessing services offered by the container [6, p. 13]. Furthermore, the control endpoint needs to provide self-description of the hardware, state and features of the device with regard to the service container [6, p. 12, pp. 22-26, p. 34]. Once a connection is established, the communication is conducted over a data endpoint, that is associated with the communication channel [6, p. 13]. Data endpoints do not have to be connection-oriented though [6, p. 47], the term is generally used for endpoints that exchange data with a subnetwork.

The ISO 22510:2019 standard specifies the implementation of the endpoint abstraction on top of the protocol stack shown in figure 4 [6, p. 39]. It does not require all of the endpoints to be implemented through a single well-known port on the server-side [6, p. 46]. Instead, clients and servers announce their endpoints to communication partners, indicating where further requests or replies shall be send to [6, p. 27-33]. This is accomplished through the host protocol address information (HPAI) structure that is part of certain frames in KNXnet/IP protocol [6, pp. 27-33, p. 45]. It consists of an IPv4 address, port and whether TCP or UDP needs to be used [6, p. 45]. Naturally, one fixed port that servers listen to is required for the initiation of KNXnet/IP communication, as otherwise clients would not know how to contact them. Hence, the discovery endpoint has to be implemented through a well-known port, unlike the other endpoints. The specification requires port 3671 to be used and it shall only be reachable through UDP [6, p. 46]. All KNXnet/IP devices have to join the multicast group of the KNX installation through IGMP and listen to incoming requests on this port [6, p. 8, p. 44]. Multicast allows a client to send a single frame to all devices in the group, without knowing their individual IP addresses. This is the basis for the device discovery [6, p. 11]. Both the control endpoints and data endpoints have to be implemented with UDP and may optionally be provided through TCP [6, p. 47]. Only clients are permitted to initiate TCP connections [6, p. 40]. The server implementation may use any port for control endpoints, but it has to be the same for both UDP and TCP [6, p. 47]. If the data endpoint is used for the communication of a connection established through a control endpoint, then any port can be used by the server [6, p. 46]. Connectionless communication for routing requires multicast through UDP on port 3671 [6, p. 82]. No restrictions are imposed on the client's choice of ports in all cases.

The specification claims that the concept of endpoints would result "[…] in a big flexibility of the actual implementation using the specific host protocol" [6, p. 46]. However, this approach is significantly more complex to implement than requiring a single well-known port to be used for all endpoints on the server-side and the benefits are questionable. Hence, issues with this design choice are discussed in subsection "3.1.6 Insecurities and Design Flaws in KNXnet/IP".

The device discovery works by sending a request with UDP to the configured multicast address at port 3671, which is then delivered to the discovery endpoints of all members in the multicast group [6, p. 27, p. 46]. Contained in the client's request is the HPAI that the replies shall be sent to via unicast [6, p. 28, p. 47]. Servers send separate responses for each of their control endpoints, containing the associated HPAI under which services for the subnetwork can be reached [6, p. 12, p. 28]. The response also contains information about the device configuration and which service families, including their versions, are supported by the control endpoint [6, p. 28]. The device information include its IA, serial number, device name, MAC address, connected KNX medium for the subnetwork and whether the programming mode is enabled [6, pp. 24-26]. The latter is required for the configuration of a device's IA [100, p. 16]. ISO 22510:2019 specifies the second version of the Core service family which includes the option to request only replies from servers that are in programming mode, have a specific MAC address or support a certain service family [6, pp. 29-32]. This allows to reduce the number of replies down to the relevant devices by filtering for the criteria on the server-side [6, pp. 30-31]. Furthermore, the client can request additional details about the server, including information about the manufacturer, IAs and their status for tunneling, whether KNXnet/IP Secure is supported, and general information about the IP configuration, such as the subnet mask and default gateway [6, pp. 23-26]. Discovering devices is of particular interest for the configuration and device management with tools such as the ETS [6, p. 19].

Self-description is accomplished through point-to-point communication with a device's control endpoint to learn information about the service container and its subnetwork [6, p. 12]. It can retrieve a subset of the information from the device discovery of Core v2 [6, p. 12, p. 27]. The client has to include the HPAI to where the reply shall be sent to in its request [6, p. 33]. According to the standard it can be useful to determine if a service family is supported by the control endpoint before attempting to establish a connection for it, in order to avoid trial and error [6, pp. 12-13, p. 11]. It is also more efficient than device discovery when the intended communication partner is already known.

KNXnet/IP implements the creation of communication channels in the application layer on top of TCP or UDP [6, p. 13, p. 47]. This is the basis for other services, such as tunneling or device management [6, p. 13]. Connection-oriented communication allows to logically separate the interaction with different clients, manage their access to limited resources, and handle stateful communication with services. Each connection in KNXnet/IP is established for the use of one respective service type [6, p. 13]. Hence, a client has to indicate which service it intends to use in the initial connect request sent to a server's control endpoint [6, p. 34]. The subnetwork for which the service is accessed and its associated control endpoint is known from the device discovery. The client's request also contains its control and data endpoint HPAIs [6, p. 35]. The control endpoint is used for managing the connection, whereas the data endpoint is used for the communication within the connection [6, p. 13]. On reception, the server can decide whether to accept the request and send a reply to the client's control endpoint [6, p. 35]. The server's response contains a status code that shows whether the connection was established successfully, the ID of the new communication channel and the allocated data endpoint for it [6, pp. 35-36]. The client can then start to use the communication channel, to interact with the service through the data endpoint. Each frame in the connection has a sequence counter, initially starting at zero [6, p. 15]. It is incremented with every frame sent, for each communication channel respectively, if UDP is used on the transport layer [6, p. 15]. In case frames arrive out of order, they could be discarded if their sequence number is lower than the expected value [6, p. 68]. The protocol allows to check the status of a connection [6, p. 36]. This can be used to perform a “heartbeat” check, ensuring it is still alive, in particular when a client uses UDP [6, p. 15]. A single TCP connection may contain multiple KNXnet/IP connections [6, p. 42]. Regularly, a connection is terminated by the client, although the server is also able to close the channel when an error occurs [6, p. 41].

3.1.2. Device Management

The device management service family facilitates the configuration of KNXnet/IP device with a management client, such as the ETS [6, p. 48]. A point-to-point connection has to be established on the KNXnet/IP layer for this purpose, as specified by the core service family [6, p. 34]. It is initiated through the control endpoint of the service container that is supposed to be configured. If the device provides access to more than one subnetwork, it has to implement a service container for each, which are treated as independent entities [6, p. 49].

Once the connection is established and communication commences through the negotiated data endpoints, the client is able to send a device configuration request [6, p. 50]. It wraps a common external message interface (cEMI) frame, which is a medium independent message format for KNX [101, p. 57]. It provides an interface to the KNX protocol layers of another device [101]. The KNX standard specifies the protocol stack from the physical layer to the application layer for its communication media TP, PL and RF [14], [102], [103], [97], [104], [105], [100]. This stack has to be provided for every service container to communicate with the respective subnetwork. Part of its functionality is to configure the device [100, p. 54]. KNXnet/IP leverages this through the cEMI by acting as a host protocol for its frames [101, p. 58]. In the context of device management a subset of cEMI frames may be used by the client. This includes frames to access the interface objects that represent the configuration of the KNXnet/IP server [101, pp. 100-102], [6, p. 50, p. 61]. They contain properties a client can read and write, if they are not read-only [101, pp. 100-102], [6, p. 58]. For instance, information such as the manufacturer ID, product ID, serial number, version and hardware type of the device can be retrieved [106, p. 28, p. 33, p. 39, pp. 67-68]. Writable properties include for example the assigned IAs, default gateway or IP address [6, p. 175]. Additionally, an interface object supports functions that can be called on it [101, pp. 103-104], [6, p. 58]. A list of the available interface objects and properties is provided in the “Resources” document of the KNX v2.1 standard [106] that is extended by the ISO 22510:2019 standard [6, pp. 175-177]. A client can reboot the server through cEMI [101, p. 105], [6, p. 58]. Changes to the configuration are required to take effect after a restart of the server and an additional 30 s have past

[6, p. 50].

Within the device management connection, the server implicitly prepares a connection on the transport layer for cEMI, giving the client access to server's application layer [6, p. 58], [101, pp. 96-97]. It may be required to perform the insecure authorization procedure analyzed by Antonini et al. [17] before commands can be issued, if it is enabled [100, pp. 89-90]. The application layer permits operations such as resetting the device to the factory default state [100, p. 47], reading and writing the memory of the communication and application controller [100, pp. 71-76, pp. 79-83], and changing the keys for the authorization as well as their permission levels [100, pp. 91-92].

The ETS can fully configure the KNXnet/IP device through a device management connection, including the application running on the device and its parameters [6, p. 49].

3.1.3. Tunneling

The tunneling service family provides the ability for a KNXnet/IP client to create a tunnel into a subnetwork through a KNXnet/IP server. It is used for device configuration and diagnostics inside the subnetwork, in particular by the ETS [6, p. 64]. Similar to the device management, it requires a KNXnet/IP connection to be established, as specified by the core service family [6, p. 34]. When the client creates the connection for the purpose of tunneling, it has to choose one of the three available modes: data link layer, cEMI raw mode and busmonitor [6, p. 71]. Every device that supports the tunneling service family has to implement the data link layer mode, the others are optional [6, pp. 65-67]. The modes will be explained later in this section. Besides the tunneling mode, the client may optionally indicate which IA of the KNXnet/IP server it wants to use for the tunnel [6, p. 71]. The server has an IA that identifies it in the KNX installation as well as additional IAs that can be used for tunneling [6, p. 65, p. 176]. Clients can retrieve the server's IAs and their availability through an extended search request from the device discovery of the core service family [6, pp. 76-77]. If the server accepts the connection request, the positive reply contains the IA assigned to the created tunnel [6, p. 71-72]. If no more unassigned addresses are available for tunneling the server will refuse, as the IA must be unique within the installation [107, p. 5]. Frames from the KNXnet/IP client send through the tunnel into the subnetwork appear to logically originate from a device in the subnetwork with the tunnel's IA [6, p. 65].

In the data link layer mode the KNXnet/IP client sends tunneling requests that contain cEMI frames for the data link layer [6, p. 64]. This is the lowest layer with a medium independent frame format [101, p. 16]. The cEMI frame itself may wrap frames from the higher layers, starting with the network layer and up to the application layer [101, p. 75], [104, p. 6], [105, p. 6], [100, p. 8]. The KNXnet/IP server passes these in the correct, medium specific format into the subnetwork. Acknowledgments are sent by the KNXnet/IP server, not the client [6, p. 65]. Frames from the subnetwork are only passed to the client if the address the IA of the tunnel or a group that the IA is in [6, p. 65]. If the KNXnet/IP server returns its own IA during the connection establishment and uses this for a tunnel, then no management of it is permitted through the tunnel or from the subnetwork [6, p. 65]. Furthermore, if the KNXnet/IP server is a router, then it may not use its own IA and has to use one of the additional IA meant for tunneling [6, p. 65].

In the raw mode the KNXnet/IP client has full control over the frame format and content that is sent into the subnetwork [6, p. 65], [101, p. 92]. The KNXnet/IP server passes all frames received from the subnetwork to the client. It does not acknowledge received frames. This mode is not allowed when the KNXnet/IP server is a router [6, p. 67]. It is intended for testing or diagnostics [101, p. 91].

When the busmonitor mode is active, the KNXnet/IP server passes all received frames from the data link layer upwards to the client [6, p. 65], [101, p. 93]. The KNXnet/IP client is cannot send frames into the subnetwork [6, p. 65]. There may only be one tunnel per subnetwork and no other services may be provided for it [6, p. 67]. Since this requirement would disable routing, busmonitor mode may not be used by KNXnet/IP routers [6, p. 67].

Through the described tunneling modes a KNXnet/IP client, such as the ETS, is able to send both arbitrary frames into the subnetwork and monitor the communication within in its entirety. Hence, a client can use tunneling to modify the configuration of KNX devices in the subnetwork similar to how device management works for KNXnet/IP devices. In the same way it also possible to prepare frames that control the state of actuators.

3.1.4. Routing

The routing service family enables the routing of frames in the KNX installation over KNXnet/IP. The KNXnet/IP routers are required to implement it, in order to provide similar functionality to line- and backbone couplers used for TP [6, p. 8, p. 78]. When a KNX device in a subnetwork attempts to send a KNX frame over TP, all devices connected to the same line are able to receive it. However, if the target is located in another line or area, it has to be passed further along. Line couplers are used to connect different lines, for instance to the main line of their area. They can pass the frame to the line higher up in the hierarchy if the target is known not to be located within the subnetwork. Thus a frame can reach other line couplers, for example those connected to the main line. Each receiving line coupler can then decide whether the frame was meant for their subnetwork and filter accordingly. If a KNX installation uses IP as medium alongside TP and the target of a frame is not located within the same area, it will eventually reach the KNXnet/IP router connecting the area of the sender to the IP backbone. All KNXnet/IP routers of an installation are part of a multicast group which they join through IGMP [6, p. 82]. They are required to implement a data endpoint for the routing service that is accessible through port 3671 [6, p. 47, p. 82]. Frames received from the subnetwork that require routing are transferred as a cEMI frame, containing the information from the data link layer and upwards, wrapped in a routing indication frame [6, p. 90, p. 92]. The KNXnet/IP router sends the resulting frame via UDP to all members of the multicast group. Like line- and backbone couplers, the recipients of the multicast filter incoming routing frames and pass matching ones into their subnetwork [6, pp. 88-89]. The cEMI frames contain a counter for the number of retransmissions that have occurred [101, p. 74]. Each KNXnet/IP router has a configurable limit for the retransmissions after which a frame is dropped [6, p. 177]. In principle the frame should eventually reach its destination, unless KNXnet/IP routers have to drop frames due to overflowing queues or exceeded retransmission limits. UDP does not provide reliable transport [6, p. 2]. The service provides lost message indication and flow control to address the issue of overflowing queues [6, pp. 83-84]. KNXnet/IP clients can send routing frames as well, which are handled in the same manner as those sent by another KNXnet/IP router.

Additionally, ISO 22510:2019 specifies the system broadcast, which is supposed to reach every device in the KNX installation [6, p. 85]. Unlike the regular routing indication frames, the system broadcast is not supposed to be filtered. All routers pass its valid frames into their subnetwork if they support it [6, p. 86]. The system broadcast permits to retrieve the serial number of devices, synchronize state required for Data Secure, send encrypted and authenticated Data Secure frames and configuring the IP address of the routing multicast group [6, p. 87]. If the latter is conducted through Data Secure, then it may also change a cryptographic key used by KNXnet/IP Secure to protect multicast communication or disable secure communication altogether [6, pp. 87-88]. Since the system broadcast relies on Data Secure to ensure confidentiality and authentication for security critical configuration and does not permit to wrap the frames in KNXnet/IP Secure [6, p. 85], it is not further analyzed as Data Secure is not within the scope of this work.

3.1.5. Remote Diagnosis and Configuration

The remote diagnosis and configuration service family is intended to be used by the ETS when a device is not reachable through a unicast connection or device discovery is not working properly [6, p. 95]. This situation can occur when required information about the device are unknown or if the device is misconfigured. Requests are sent through UDP to the multicast group [6, p. 95]. Broadcasting may optionally be used if multicast fails [6, p. 95].

A KNXnet/IP client may send a diagnostic request to retrieve information about the current configuration of a device in the KNX installation. For this purpose the frame includes a selector that indicates which device is supposed to answer the request [6, p. 96]. This can either be a specific MAC address or an active programming mode [6, p. 100]. Additionally, the frame contains the HPAI for the data endpoint replies shall be sent to [6, p. 96]. The KNXnet/IP server that matches the selector sends its reply to the data endpoint containing information about the current configuration of the device [6, p. 95]. Unlike device discovery, it does not restrict the reply to specific device information. It is supposed to send them in their entirety [6, p. 95], [6, p. 23].

A configuration request can be sent by a KNXnet/IP client in order to change the configuration of a specific KNXnet/IP server. Similar to the diagnostics request it contains a selector and HPAI for its data endpoint [6, p. 97]. Furthermore, the frame includes entries for every property that is supposed to be changed [6, p. 97]. The receiving device that matches the selector is supposed to apply the submitted values, if the respective property it is not read-only [6, p. 97]. KNXnet/IP server acknowledge the changes with a reply that contains its current configuration [6, p. 95].

A KNXnet/IP client may also request a restart or factory reset of KNXnet/IP device [6, p. 95]. This frame uses a selector to identify the intended recipient as well [6, p. 98].

3.1.6. Insecurities and Design Flaws in KNXnet/IP

The protocol design of KNXnet/IP is lacking in security because it does not provide confidentiality, authentication of frames and their origin, or properly implement authorization. The lack of confidentiality enables an adversary to eavesdrop on the communication. Missing authentication of frames allows to manipulate their content without the recipient noticing. When the origin of the frame is not authenticated the recipient cannot verify which party has sent it. Additionally, KNXnet/IP itself does to implement authorization. Therefore, any client, including an adversary, can use the services it provides. The only time authorization may be required is when accessing the KNX application layer through the cEMI [100, p. 89]. In this case KNXnet/IP relies on the existing authorization feature provided by the KNX stack. It is insecure as the required key is transmitted as plaintext [16], [17], [100, p. 89]. In contrast to the missing security in the protocol design, KNXnet/IP defines extensive services for the ETS to manage and monitor the devices. However, due to the lack of access control the adversary can use them too, risking both passive and active attacks against the KNX installation. The previous sections have summarized the relevant parts of the protocol specification from the point of view of their intended use. This section highlights problems from the perspective of an attacker, which would need to be addressed by KNXnet/IP Secure.

The device discovery and self-description from the core service family provide information about each KNXnet/IP device, which may be enough for an adversary to identify the exact model and make. Information that can be retrieved include the manufacturer ID [6, p. 27], type of device (e.g. KNXnet/IP router) [6, p. 26], [106, p. 22], serial number and display name ("friendly name") [6, p. 25]. If certain KNXnet/IP devices are known to have vulnerabilities, an attacker could use these information to identify and pick specific targets. For a defense-in-depth approach the amount of information disclosed to an unauthorized client should be limited to a minimum. The core service family also specifies the creation of connections which are required to access the device management and tunneling services [6, p. 34]. The handshake should have included an AKE to establish session keys for the encryption, authenticate the client and determine what services it is authorized to use within the connection. Without these measurements an attacker is able to create a connection to use these services as well. Similar challenges arise for the connectionless communication through multicast for the core, routing, and remote diagnosis and configuration services. An adversary should not be able to join the group and issues requests, but KNXnet/IP's use of IGMP does not provide any access control.

The device management connection gives access to the interface objects and associated properties which would allow an adversary to both read and modify the configuration of a device [6, p. 61]. Information that can be gathered about it are more detailed than the replies of self-description and device discovery, thus they would allow to identify the specific product being used. Misconfiguration of the KNX devices through the interface objects can hinder the operation of the installation. After passing authorization the device management connection can use all services provided by the KNX application layer through cEMI. As previously explained, the authorization is insecure as the respective frames are sent unencrypted [6, p. 89]. Hence, an adversary could be able to intercept these transmissions, for instance from a legitimate client like the ETS, and gain knowledge of the keys. Consequentially, the attacker would then be able to authorize itself with the obtained key material. This flaw was described by Granzer et al. [16] and practically demonstrated by Antonini et al. [17]. The adversary could also try an online attack against the system by attempting to brute force the 32 bit key space. Antonini et al. [17] tested this for devices located in the subnetwork, connected to TP. They argued that the limited data rate caused a slow response by the actuators of approximately a second per request and therefore concluded that an online attack would not be feasible [17]. However, when the target is the KNXnet/IP router and not a device in the subnetwork, the assumption about the strongly limited data rate is no longer true. Hence, an online attack through a device management connection could be possible, but the feasibility depends on the time required by the target device to process the frame and send a reply. If an adversary is able to gain access to the application layer through either method with a sufficient access level, they can read and write the memory of the remote communication controller and remote application controller [100, pp. 71-76, pp. 79-87]. Thus, the attacker would be able to perform the same operations as the ETS. Whether the writable memory of the application controller configures the application program or alters it, appears to depend on the specific implementation of the device. The specification states that "[...] Application Device Management may also influence directly or indirectly the application program" and "[...] this has to be defined individually for each device" [100, p. 79]. The security impact may therefore vary.

A tunneling connection permits interacting with the subnetwork connected to a KNXnet/IP device [6, p. 64]. In the data link layer mode an adversary can configure devices in the subnetwork, similar to the device management. The research by Molina [18] demonstrates that actuators can be remote controlled through a tunneling connection. While the given example [18, pp. 7-8] uses lighting to demonstrate the problem, more security sensitive use cases are equally affected when using KNXnet/IP. In raw mode it is possible to send arbitrary byte sequences into the subnetwork [6, p. 65], [101, pp. 91-92]. This could be useful for an adversary because they are not restricted to specific frame formats predefined by the cEMI. An adversary can thus construct unusual frame formats that are not conforming with the standard, which could help to exploit bugs in the implementation of the KNX layers, if they exist. The busmonitor mode allows to inspect all communication in the subnetwork [6, p. 67], [101, p. 93]. Hence, it can be used by an attacker to eavesdrop on it.

The routing in KNXnet/IP relies on a multicast group that the routers join through IGMP [6, p. 82]. Since there is no access control specified, an adversary can join the group as well. Frames that are being routed over the IP backbone can be eavesdropped on and the attacker can send its own frames through multicast.

The remote diagnosis and configuration service family can be used to perform basic configuration of devices through multicast or broadcast with KNXnet/IP [6, p. 95]. The risk is similar to device management, since the lack of access control permits an adversary to read and alter parts of the configuration through this approach as well. Any client can also trigger a reboot or factory reset of a KNXnet/IP device [6, p. 95]. The latter would be an effective tool for a denial of service attack against the KNX installation.

The design of KNXnet/IP was criticized by Granzer et al. [25] in 2009 for relying on "security by obscurity". The most recent version of the publicly accessible KNX standard v2.1 contains an overview document about KNXnet/IP, which was written in 2013 [15]. This version of the document still contains a chapter about security considerations that greatly misjudges the risks as well as the need for secure communication. More specifically, the document concludes: "It is quite unlikely that legitimate users of a network would have the means to intercept, decipher, and then tamper with the KNXnet/IP without excessive study of the KNX Specifications. Thus the remaining security threat is considered to be very low and does not justify mandating encryption, which would require considerable computing resources." [15, p. 12]. The KNX Association has been slow in the adoption of security best-practice in the past. Hence, it has to be thoroughly analyzed whether such issues still exist in KNXnet/IP Secure and the ETS.

Besides the security problems, there is also a more general criticism of the KNXnet/IP specification. It permits manufacturers to choose whether TCP is supported [6, p. 7]. The control and data endpoints can be implemented on the same port, but they are not required to [6, p. 46]. As previously cited, the specification claims that the concept of endpoints would provide flexibility for the implementation [6, p. 46]. However, it complicates the protocol design because KNXnet/IP clients and servers cannot generally be sure how their communication partners chose to implement the protocol and what optional aspects they support. For instance, the HPAI is necessary because the devices have to announce under which IP, port and protocol their endpoints can be reached. If KNXnet/IP had made it mandatory for all endpoints of the KNXnet/IP servers to be implemented through a single well-known port and picked one transport protocol, then this would not have been required. Even when a server has more than one control endpoint, they could have been distinguished through a numerical identifier. Furthermore, supporting the tunneling and device management services through TCP and UDP, with UDP being mandatory, requires transport layer dependent implementations of KNXnet/IP. UDP does not provide reliable transport, hence the implementation of the services have to handle sequence numbers, acknowledgments and retries [6, p. 50, pp. 67-68]. For TCP this is not needed. These examples highlight the difficulty to ensure interoperability in the protocol design when certain aspects are optional or only roughly specified, permitting manufacturers to implement it in different ways. While there may be good reasons for the design choices made for KNXnet/IP, the claimed flexibility and options provided to the manufacturers appear to have significantly increased the complexity of the specification. Generally, it is not beneficial for the avoidance of bugs when a specification and thus implementation is more complex than technically necessary. Furthermore, ambiguities in the specification can result in implementations that technically fulfill the requirements but solve them in a suboptimal or even harmful way, because the developers had to guess what the intended solution was meant to be. This approach towards specifying a protocol should not be used for a cryptographic protocol such as KNXnet/IP Secure. If the specification is not unambiguous and strict in its requirements, it either leaves room for misinterpretation or permits options that may have a negative impact on security properties. Therefore, the specification of KNXnet/IP Secure needs to be checked for such issues.

3.2. Cryptography

Understanding the specification of KNXnet/IP Secure and its analysis in this work requires knowledge about the cryptography used within. This section introduces the relevant cryptographic primitives, the definition of security properties that KNXnet/IP Secure can be evaluated against and the eCK-PFS model for a formal analysis. The research is concerned with the correct application of cryptography, not an analysis of the cryptographic primitives themselves. Hence, this section limits itself to definitions of cryptographic primitives that are of interest due to their potential for misapplication and resulting impact on the security.

3.2.1. CCM Cipher Mode

KNXnet/IP Secure uses the CCM block cipher mode in combination with AES-128 for all symmetric encryption [6, p. 102]. It provides AEAD even though it is an authenticate-then-encrypt block cipher mode [38]. While the generic composition of authenticate-then-encrypt is not secure [33], [34], this particular construction has a security proof by Jonsson [35]. It combines a CBC-MAC for authentication with the CTR cipher mode for encryption [2]. The encryption and decryption algorithm are shown in figure 5 and 6 respectively. Both algorithms make use of two function, the formatting function β and the counter generation function π. The former splits the input into blocks over which the CBC-MAC is calculated, while the latter generates the counter blocks that are used to generate the key stream. The NIST Special Publication 800-38C that standardizes CCM suggests implementations for them, but they are not mandatory [3]. Therefore, the standard permits the use of custom formatting and counter generation functions. This decision has been criticized by Rogaway [38]. Based on the security proof [35], the standard defines conditions that the functions β and π have to satisfy, as shown in definition 3.3. Additionally, it is required that the nonces N are non-repeating under a given key as described in definition 3.2. The conditions are important because the security proof relies on the assumption that they hold true.

Thus, any violations of them may affect the security properties.

Definition 3.1 (CCM notation). Below is the definition of the notation used in the CCM encryption and decryption algorithms.

k	Secret symmetric key k used for both the encryption of the plaintext P and decryption of the ciphertext C.
N	Nonce N which needs to be non-repeating for all invocations under a given key k [35].
P	Plaintext P that is encrypted/decrypted and authenticated with CCM.
A	Associated data A that is authenticated but not encrypted/decrypted with CCM.
B_i	The i-th block B_i created with the formatting function β as input for the CBC-MAC computation.
Ctr_i	The i-th counter block Ctr_i created with the counter generation function π as input for the keystream computation.
T	Tag T that authenticates the content of the ciphertext C.
T_{len}	Tag length T_{len} measured in bits.
S	Keystream S for the encryption/decryption.
C	Ciphertext C encrypted and authenticated result of CCM. It includes the encrypted tag T.
$x \leftarrow y$	Assign variable x the value of y, which can be another variable, the result of a function or an expression.
$x \oplus y$	Exclusive or operator that is applied bitwise on the operands.
$x \| y$	Concatenation of the bits in x and y.
$\text{len}(x)$	Length of x in bits
β	Formatting function that creates the blocks $B_0, B_1, ..., B_r$ for the CBC-MAC computation. The definition is not standardized, but it is supposed to fulfill the requirements listed in definition 3.3.
π	Counter generation function that creates the $Ctr_0, Ctr_1, ..., Ctr_m$ from which the keystream S is computed. The definition is not standardized, but it is supposed to fulfill the requirements listed in definition 3.3.
$\text{MSB}_x(y)$	Most significant x number of bits from y.
$\text{LSB}_x(y)$	Least significant x number of bits from y.
$\text{Ciph}(k, I)$	Block cipher *Ciph* applied to an input I under the key k. In case of CCM the block cipher is AES-128 [3], [37].

```
CCM-ENC($k, \pi, \beta, T_{len}, N, P, A$)
1 :  $B_0, B_1, ..., B_r \leftarrow \beta(N, A, P)$
2 :  $Y_0 \leftarrow \mathrm{Ciph}(k, B_0)$
3 :  for $i \leftarrow 1$ to $r$ do
4 :    $Y_i \leftarrow \mathrm{Ciph}(k, B_i \oplus Y_{i-1})$
5 :  $T \leftarrow \mathrm{MSB}_{T_{len}}(Y_r)$
6 :  $Ctr_0, Ctr_1, ..., Ctr_m \leftarrow \pi(N, m)$ where $m = \lceil \mathrm{len}(P)/128 \rceil$
7 :  for $j \leftarrow 0$ to $m$ do
8 :    $S_j \leftarrow \mathrm{Ciph}(k, Ctr_j)$
9 :  $S \leftarrow S_1 \| S_2 \| ... \| S_m$
10 : return $C \leftarrow (P \oplus \mathrm{MSB}_{\mathrm{len}(P)}(S)) \| (T \oplus \mathrm{MSB}_{T_{len}}(S_0))$
```

Figure 5: CCM encryption, based on [3]
with canonical formatting and counter generation functions

```
CCM-DEC($k, \pi, \beta, T_{len}, N, C, A$)
1 :  if $\mathrm{len}(C) \leq T_{len}$ then
2 :    return $INVALID$
3 :  $Ctr_0, Ctr_1, ..., Ctr_m \leftarrow \pi(N, m)$ where $m = \lceil (\mathrm{len}(C) - T_{len})/128 \rceil$
4 :  for $j \leftarrow 0$ to $m$ do
5 :    $S_j \leftarrow \mathrm{Ciph}(k, Ctr_j)$
6 :  $S \leftarrow S_1 \| S_2 \| ... \| S_m$
7 :  $P \leftarrow \mathrm{MSB}_{\mathrm{len}(C)-T_{len}}(C) \oplus \mathrm{MSB}_{\mathrm{len}(C)-T_{len}}(S)$
8 :  $T \leftarrow \mathrm{LSB}_{T_{len}}(C) \oplus \mathrm{MSB}_{T_{len}}(S_0)$
9 :  if $N, A$ or $P$ not valid [3, p. 8] then
10 :   return $INVALID$
11 : $B_0, B_1, ..., B_r \leftarrow \beta(N, A, P)$
12 : $Y_0 \leftarrow \mathrm{Ciph}(k, B_0)$
13 : for $i \leftarrow 1$ to $r$ do
14 :   $Y_i \leftarrow \mathrm{Ciph}(k, B_i \oplus Y_{i-1})$
15 : if $T \neq \mathrm{MSB}_{T_{len}}(Y_r)$ then
16 :   return $INVALID$
17 : return $P$
```

Figure 6: CCM decryption, based on [3]
with canonical formatting and counter generation functions

Definition 3.2 (**Non-repeating nonce** [3])**.** "A bit string called the nonce, denoted N, is assigned to the data pair to be protected, i.e., the payload and its associated data. The nonce shall be non-repeating in the sense that any two distinct data pairs to be protected by CCM during the lifetime of the key shall be assigned distinct nonces."

Definition 3.3 (Requirements for input formatting [3]**).**

1. The first block, B_0, uniquely determines the nonce N.
2. The formatted data uniquely determines P and A; moreover, if (N, P, A) and (N, P', A') are distinct input triples whose formatting is $B_0, B_1, ..., B_r$ and $B'_0, B'_1, ..., B'_{r'}$, then B_i is distinct from B'_i for some index i such that $i \leq r$ and $i \leq r'$.
3. The first block, B_0, is distinct from any counter blocks that are used across all invocations of CCM under the key.

3.2.2. Elliptic Curve Cryptography and Curve25519

KNXnet/IP Secure specifies an AKE for its unicast connections that uses ECDH to determine the session key. It relies on Curve25519, which is an elliptic curve developed by Daniel J. Bernstein [42], [6, p. 121]. Hence, this section provides a brief overview of elliptic curves and their application in cryptography as well as Curve25519. Since KNXnet/IP Secure is the focus of this work and not ECC, the reader is referred to the works of Hankerson et al. [32], Boneh and Shoup [5], Bernstein [42] and BSI TR-03111 [46] for a detailed explanation.

Definition 3.4 (Elliptic curve). Let p be a prime number and F_p the finite field of integers modulo p [32, p. 13], [5, p. 613]. A Montgomery curve E, defined over F_p, is an equation

$$By^2 = x^3 + Ax^2 + x \tag{1}$$

for some $A, B \in \mathbb{F}_p$ where $B(A^2 - 4) \neq 0$ [5, p. 615].

Definition 3.5 (Points on the curve). Let E be a Montgomery curve defined over F_p and $e \geq 1$. Let ∞ be the point at infinity, which is the identity element [5, p. 614]. The set of points on the curve is defined as [42, p. 226], [5, p. 613]:

$$E(\mathbb{F}_{p^e}) = \{\infty\} \cup \{(x, y) \in \mathbb{F}_{p^e} : By^2 = x^3 + Ax^2 + x\} \tag{2}$$

When $e = 1$ the points of the curve are defined over the base field and if $e > 1$ they are defined over the respective extension field [5, p. 613].

Definition 3.6 (Group law). $E(\mathbb{F}_{p^e})$ is an abelian group [5, p. 614]. The following rules apply for all affine points $P = (x, y)$ with $P \in E(\mathbb{F}_{p^e})$ in Montgomery curves

[108], [5, p. 614]:

$$-\infty = \infty \tag{3}$$
$$\infty + \infty = \infty \tag{4}$$
$$\infty + P = P + \infty = P \tag{5}$$
$$-(x, y) = (x, -y) \tag{6}$$
$$(x, y) + (x, -y) = \infty \tag{7}$$

Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ with $P, Q \in E(\mathbb{F}_{p^e})$ and $P + Q = (x_3, y_3)$.
If $x_1 \neq x_2$ then

$$(x_1, y_1) + (x_2, y_2) = -(x_3, y_1 + \lambda(x_3 - x_1)) \tag{8}$$
$$\lambda = (y_2 - y_1)/(x_2 - x_1) \tag{9}$$
$$x_3 = B\lambda^2 - A - x_1 - x_2 \tag{10}$$

If $x_1 = x_2$, $y_1 = y_2$ and $y_1 \neq 0$ then

$$(x_1, y_1) + (x_1, y_1) = -(x_3, y_1 + \lambda(x_3 - x_1)) \tag{11}$$
$$\lambda = (3x_1^2 + 2Ax_1 + 1)/(2By_1) \tag{12}$$
$$x_3 = B\lambda^2 - A - 2x_1 \tag{13}$$

The addition of a point P to itself for α times is defined as $\alpha P = (\alpha - 1)P + P$ for any positive integer α [5, p. 614].

The multiplication of a point with a scalar can be efficiently solved with the Montgomery ladder algorithm [109] using only the x-coordinate of the point. Additionally, scalar multiplication can be implemented as a constant-time ladder algorithm [108], which is useful to reduce the chance of timing attacks. Elliptic curve cryptography build on top of the arithmetic from definition 3.6 requires the intractability of the elliptic curve discrete logarithm problem (ECDLP), elliptic curve computational Diffie-Hellman problem (ECCDHP) and elliptic curve decision Diffie-Hellman problem (ECDDHP).

Definition 3.7 (**ECDLP** based on [32, p. 153], [5, p. 615])**.** Let E be an elliptic curve defined over $\mathbb{F}_{p^e}$ with $e \geq 1$. Given point $P \in E(\mathbb{F}_{p^e})$ of prime order q, so that $qP = \infty$, and a point $Q = \alpha P$, determine the integer $\alpha \in [0, q - 1]$.

Definition 3.8 (**ECCDHP** based on [32, p. 171])**.** Let E be an elliptic curve defined over $\mathbb{F}_{p^e}$ with $e \geq 1$. Given point $P \in E(\mathbb{F}_{p^e})$ of prime order q, and points $Q = \alpha P$, $R = \beta P$, determine point $S = \alpha\beta P$.

Definition 3.9 (**ECDDHP** based on [32, pp. 171-172])**.** Let E be an elliptic curve defined over $\mathbb{F}_{p^e}$ with $e \geq 1$. Given point $P \in E(\mathbb{F}_{p^e})$ of prime order q, so that $qP = \infty$, and points $Q = \alpha P$, $R = \beta P$, $S = \gamma P$, determine whether $S = \alpha\beta P$ or equivalently, whether $\gamma \equiv \alpha\beta \pmod{q}$.

Whether the ECDLP and related problems are hard to solve depends on the domain parameters of the specific curve used for ECC. Besides the coefficients for the curve E, a base point P with prime order q has to be selected. The base point creates a cyclic subgroup of $E(\mathbb{F}_{p^e})$ that is defined over the set $\{\infty, P, 2P, 3P, ..., (q-1)P\}$ [32, p. 13], where q is the smallest scalar factor for which $qP = \infty$. Furthermore, $|E(\mathbb{F}_{p^e})|$ is the number of points on the curve, which can be calculated efficiently [5, p. 614]. The cofactor of a curve is $h = |E(\mathbb{F}_{p^e})|/q$ [32, p. 172].

Based on the work of Victor Shoup [110], it is known that the lower bound of the runtime complexity for solving the Diffie-Hellman problem in generic subgroups of prime order q is $\Omega(\sqrt{q})$. This lower bound also applies to the ECCDHP and ECDDHP [32, p. 171-172]. In order to prevent successful attacks against the elliptic curve by solving the ECDLP with techniques such as Pollard's rho, which has a runtime complexity of $\mathcal{O}(\sqrt{q})$, the q should be sufficiently large [32, p. 154, p. 172]. However, there are attacks that can solve the ECDLP faster if the elliptic curve has specific properties [5, p. 615]. For instance, if $|E(\mathbb{F}_p)| = p$ it can be solved in polynomial time [5, p. 615], [32, pp. 168-169]. For a detailed overview of techniques that improve on the runtime complexity of the generic solution for the ECDLP, see [111]. All these attacks need to be considered when designing a elliptic curve for a cryptographic system in order to ensure an adversary cannot solve the ECDLP. If the domain parameters are chosen accordingly, the resulting elliptic curve can be used for asymmetric cryptography. A private key is selected as a uniformly random $\alpha \in [1, q-1]$ and the public key Q is calculated through $Q = \alpha P$. Since the assumption is that an adversary cannot solve the ECDLP, they are unable to determine the private key from the public key. On this basis the ECDH key exchange can be implemented, as shown in figure 7.

$$
\begin{array}{lcl}
\textbf{Client} & & \textbf{Server} \\
\hline
\alpha \overset{R}{\leftarrow} \{1, 2, ..., q-1\} & & \\
Q \leftarrow \alpha P & & \\
 & \xrightarrow{\quad Q \quad} & \\
 & & \beta \overset{R}{\leftarrow} \{1, 2, ..., q-1\} \\
 & & R \leftarrow \beta P \\
 & & S \leftarrow \beta Q \\
 & \xleftarrow{\quad R \quad} & \\
S \leftarrow \alpha R & &
\end{array}
$$

Figure 7: ECDH

The ECDH allows the two parties to arrive at a shared secret S due to the commutativity of the operations in $\mathbb{F}_{p^e}$. Hence $\alpha R = \alpha(\beta P)$ is equivalent to $\beta Q = \beta(\alpha P)$. Same as the Diffie-Hellman key exchange, the ECDH only provide confidentiality and not authentication. Therefore, it is susceptible to MitM attacks. However, it can be used as the basis to construct an AKE. The design of the ECDH implementation needs to consider small subgroup and invalid curve attacks as they could be used to gain information about the private key [32, pp. 181-182]. One way to prevent them is to perform input validation. For example the invalid curve attack can be avoided by checking that the received point lies on the curve [32, p. 182].

Definition 3.10 (Curve25519 [42]**).** The Curve25519 is a Montgomery curve specified over $E(p^2)$ with $p = 2^{255} - 19$ by the equation

$$y^2 = x^3 + 486662x^2 + x \tag{14}$$

The base point is defined as $(9, ...)$ with a positive y-coordinate and its order is a prime

$$2^{252} + 27742317777372353535851937790883648493 \tag{15}$$

Bernstein [42] developed both an elliptic curve for the application in cryptography, as shown in definition 3.10, and a function for the scalar multiplication for ECDH. This thesis refers to the former as Curve25519 and the latter as X25519, following the nomenclature of RFC 7748 [7]. The design of both the curve and the function were evaluated for their security properties by Bernstein [42]. In particular the attacks with Pollard's rho and kangaroo method, batch discrete logarithms and small-subgroups were considered [42]. Bernstein concludes that the "security level will remain comfortable for the foreseeable future" [42] with regard to Pollard's rho and kangaroo attacks. Given the runtime complexity of Pollard's rho of $\mathcal{O}(\sqrt{q})$ and the large prime order of Curve25519 this claim appears to be accurate. Since the base point has prime order, the Pohlig-Hellmann method cannot be used to solve the ECDLP faster [42], [32, p. 155]. Additionally, the Curve25519 is supposed to provide secure twist [42], thus preventing certain types of small-subgroup attacks. Since the security assertions by Bernstein do not appear to be contested in scientific publications, Curve25519 and X25519 are treated as secure primitives in this thesis.

The function X25519(n, q) implements the scalar multiplication using the Montgomery ladder algorithm [7], [42]. It takes two 256 bit values as arguments, the private key n and the x-coordinate of a point, q. In Bernstein [42] the values are defined as $n \in 2^{254} + 8\{0, 1, 2, 3, ..., 2^{251} - 1\}$ and $q \in \{0, 1, ..., 2^{256} - 1\}$. However, the function as specified by RFC 7748 [7] permits $n \in \{0, 1, ..., 2^{256} - 1\}$ as input and transforms the otherwise forbidden values to suitable scalars internally. This is known as "private key clamping" [112, p. 86], which maps the input to an $n' \in 2^{254} + 8\{0, 1, 2, 3, ..., 2^{251} - 1\}$. X25519 returns the result of the scalar multiplication as an x-coordinate encoded in 256 bit. The function can be used to implement the ECDH key exchange as shown in figure 8. Both the RFC 7748 [7] and the publication by Bernstein [42] require the shared secret to be passed through a key derivation function to generate a symmetric key. X25519 does not ensure contributory behavior, meaning that it is not guaranteed that the private keys of both parties contribute to the resulting shared secret, due to low order points being permitted as inputs [7]. An x-coordinate (public key) with this property causes the shared secret to be zero. Such inputs may optionally be rejected according to RFC 7748 [7]. This part of the X25519 design appears to be the subject of discussions among cryptographers [113], [114], [112, p. 86].

$$\begin{array}{lcl}
\textbf{Client} & & \textbf{Server} \\
\hline
\alpha \xleftarrow{R} \{0, 1, ..., 2^{256} - 1\} & & \\
x_1 \leftarrow \mathrm{X25519}(\alpha, 9) & & \\
 & \xrightarrow{\quad x_1 \quad} & \\
 & & \beta \xleftarrow{R} \{0, 1, ..., 2^{256} - 1\} \\
 & & x_2 \leftarrow \mathrm{X25519}(\beta, 9) \\
 & & x_3 \leftarrow \mathrm{X25519}(\beta, x_1) \\
 & \xleftarrow{\quad x_2 \quad} & \\
x_3 \leftarrow \mathrm{X25519}(\alpha, x_2) & &
\end{array}$$

Figure 8: ECDH with X25519, based on [7]

3.2.3. Key Derivation with PBKDF2-HMAC-SHA-256

KNXnet/IP Secure specifies an authentication scheme for unicast connections, which is build upon the password-based key derivation function PBKDF2 with HMAC-SHA-256 as PRF [6, p. 135]. The purpose of PBKDF2 is the generation of a key for cryptographic operations from a password, as a password string cannot be directly used as a key [8]. A description of PBKDF2 is provided in figure 9. It takes a password p, salt s, iteration count c, and derived key length dk_{len} as arguments. The p is the password that the key of length dk_{len} is derived from [8]. The salt s is a byte string that increases the possible numbers of keys that can be generated from one password. It is not meant to be secret [8], but it should be different for every password associated with an identity. This prevents the same key from being derived for two users who happen to pick the same password. As a consequence it hinders an attacker from precomputing keys for passwords [8]. For a given password p they would have to generate a key for every possible value of s. For a sufficiently large salts, it becomes infeasible to compute and store the resulting combinations. Thus, it prevents the application of dictionary attacks such as precomputed rainbow tables [115] for commonly used passwords. RFC 8018 recommends a salt of at least 64 bits [8].

The salt increases the difficulty of both online and offline attacks. If a system applies client-side hashing for its authentication and the adversary is unable to precompute keys for common passwords because the salt varies with each user, then they can no longer efficiently try to find accounts with weak passwords to compromise. If an adversary is able to breach the password database, where the keys and salts are stored, they would not be able to quickly identify users who picked a particular password or

```
PBKDF2($p, s, c, dk_{len}$)
1:  if $dkLen > (2^{32} - 1) \cdot h_{len}$ do
2:      return derived key too long
3:  $l \leftarrow \lceil dk_{len}/h_{len} \rceil$
4:  $r \leftarrow dk_{len} - (l - 1) \cdot h_{len}$
5:  for $i \leftarrow 1$ to $l$ do
6:      $T_i \leftarrow 0$
7:      $U_0 \leftarrow s \| \mathrm{INT}(i)$
8:      for $j \leftarrow 1$ to $c$ do
9:          $U_j \leftarrow \text{HMAC-SHA-256}(p, U_{j-1})$
10:         $T_i \leftarrow T_i \oplus U_j$
11: return $dk \leftarrow T_1 \| T_2 \| ... \| T_l\langle 0, ..., r - 1\rangle$
```

Figure 9: PBKDF2-HMAC-SHA-256, based on [8]
INT(i) is i encoded as big endian 32 bit integer and
$T_l\langle 0, ..., r - 1\rangle$ denotes the substring of bytes 0 to $r - 1$

```
HMAC-SHA-256($k, m$)
1:  $b \leftarrow 64$
2:  $opad \leftarrow \underbrace{\texttt{0x36} \| ... \| \texttt{0x36}}_{b \text{ times}}$
3:  $ipad \leftarrow \underbrace{\texttt{0x5c} \| ... \| \texttt{0x5c}}_{b \text{ times}}$
4:  if $\mathrm{len}(k) > b \cdot 8$ then
5:      $\hat{k} \leftarrow \text{SHA-256}(k)$
6:  else
7:      $\hat{k} \leftarrow \mathrm{PAD}_b(k)$
8:  $\text{SHA-256}((\hat{k} \oplus opad) \| \text{SHA-256}((\hat{k} \oplus ipad) \| m))$
```

Figure 10: HMAC-SHA-256, based on [116] [117]
SHA-256(x) as specified in [9]

selected the same password as another user. This assumes that the system does not use a constant salt. A random, per user salt thus protects against efficient attacks targeting a group of users and especially those among them that reuse weaker passwords across different systems. However, if the adversary has access to the keys and their associated salts, they could still try a dictionary or brute-force attack against a specific entry, in an attempt to retrieve the original password. The computational complexity of PBKDF2 depends on the iteration count c, which determines how often the PRF is calculated for each block of the derived key, as shown in figure 9. In order to increase the computational costs for an attacker, it should be sufficiently high. However, an increase in c also affects the computational costs for a legitimate user, thus a trade-off is necessary. NIST SP 800-132 recommends that "The number of iterations should be set as high as can be tolerated for the environment, while maintaining acceptable performance." [118]. Password-cracking techniques utilizing GPUs for highly parallelized attacks or application-specific integrated circuits (ASICs) have to be considered when choosing the iteration count.

3.2.4. Security Properties for Cryptographic Protocols

The analysis of cryptographic protocols has the goal to identify potential weaknesses that may affect its security. Such issues can take many forms, hence the definition of desirable security properties can help guide the process. This section provides informal definitions of security properties for the analysis of KNXnet/IP Secure.

Definition 3.11 (**Confidentiality**, based on [52][51, p. 2])**.** A protocol provides confidentiality, if the encrypted communication between honest parties imparts no insight about the plaintext to an adversary. This assumes that only authorized parties have knowledge of the encryption keys.

Definition 3.12 (**Data integrity**, based on [31, p. 4, p. 361][51, p. 2])**.** A protocol provides data integrity, if an adversary is not able to alter, delete or inject messages in a communication between (authorized) honest parties, without the manipulation being detected.

Definition 3.13 (**Message authentication**, based on [31, p. 25, p. 361][51, pp. 2-3])**.** A protocol provides message authentication, if a party can be verified as the origin of the message at some point in the past and data integrity is ensured.

Definition 3.14 (**Entity authentication**, based on [31, p. 24, p. 386])**.** A protocol provides entity authentication, if one party is able to verify the identity of another party and both are actively communicating, thus providing a timeliness and aliveness guarantee.

Definition 3.15 (**Mutual authentication**, based on [65, pp. 232-233])**.** A protocol provides mutual authentication when entity authentication is performed for every party involved in the communication.

Definition 3.16 (**Non-repudiation**, based on [31, p. 4])**.** A protocol provides non-repudiation, if a party is prevented from denying previous commitments or actions.

Definition 3.17 (**Resistance against replay attacks**, based on [52])**.** A protocol provides resistance against replay attacks, if a recipient of a message can only accept it once at most.

Definition 3.18 (**Perfect forward secrecy**, based on [31, p. 496][119])**.** A protocol provides perfect forward secrecy (PFS), if the long-term (private) keys of one or more parties are compromised and this does not affect the secrecy of past session keys established between honest parties.

Definition 3.19 (**Resistance to known session key attack**, based on [120]). A protocol provides resistance to known session key attacks, if the compromise of a particular session key does not give an adversary any advantage in attacking other unrelated sessions.

Definition 3.20 (**Resistance against key compromise impersonation**, based on [120][119]). A protocol is resistant against key compromise impersonation if an adversary has compromised the long-term keys of a party, but is not able to impersonate other parties to it.

Definition 3.21 (**Resistance against unknown key-share attack**, based on [50], [119]). A protocol is resistant against unknown key-share attacks (UKS) if an adversary is unable to mislead one of the honest parties performing a key agreement into believing that they are sharing a key with a different party while they are actually communicating with the honest partner.

Definition 3.22 (**Implicit key authentication**, based on [31, p. 492][50]). A protocol provides implicit key authentication if among honest parties, one is assured that no other party except one particular identified second party (and possibly additional trusted parties), may gain access to a specific key.

Definition 3.23 (**Key confirmation**, based on [31, p. 492]). A protocol provides key confirmation if one party is assured that a second, possibly unidentified, party is in possession of a particular key.

Definition 3.24 (**Explicit key authentication**, based on [31, p. 492]). A protocol provides explicit key authentication when both implicit key authentication and key confirmation hold.

Definition 3.25 (**Key freshness**, based on [31, p. 494]). A protocol provides key freshness when honest parties use a fresh, independent session key for every session.

Definition 3.26 (**Key control**, based on [31, p. 494]). A protocol provides unbiased key control if neither party involved in the key agreement can control or predict the key beforehand.

Definition 3.27 (**Identity hiding**, based on [52]). A protocol provides identity hiding if a passive adversary that observes the traffic cannot infer the long-term keys used in the handshake.

3.2.5. eCK-PFS Model

The eCK-PFS is a formal model for the security analysis of key exchange protocols [68]. This section introduces the relevant definitions, so that it can be applied to KNXnet/IP Secure in chapter "Analysis of KNXnet/IP Secure in ISO 22510:2019".

Definition 3.28 (**eCK-PFS notation**, based on [4][68])**.** Let $\mathcal{P} = \{\hat{P}_1, \hat{P}_2, ..., \hat{P}_n\}$ be a finite set of n parties' identities. Each party can execute multiple instances of a key exchange protocol, called session, concurrently. Session s at party $\hat{P}$ is denoted as the tuple $(\hat{P}, s) \in \mathcal{P} \times \mathbb{N}$. Every session s is associated with a $T_s = (s_{actor}, s_{peer}, s_{role}, s_{sent}, s_{recv}) \in \mathcal{P} \times \mathcal{P} \times \{\mathcal{I}, \mathcal{R}\} \times \{0,1\}^* \times \{0,1\}^*$. The variable s_{actor} denotes the identity of the actor and s_{peer} the intended peer of session s. Note that the actual peer does not have to be s_{peer}, it is only the party s_{actor} intends to communicate with. The values of the variables s_{peer} and s_{role} are set upon activation of the session s. The possible values for s_{role} are $\{\mathcal{I}, \mathcal{R}\}$, where $\mathcal{I}$ stand for initiator and $\mathcal{R}$ for responder. The protocol execution steps define the values of s_{sent} and s_{recv}. Messages being send or received are appended to the respective variables. A session can only be activated once.

Definition 3.29 (**Adversary capabilities**, based on [4][68])**.** The adversary $\mathcal{A}$ is modeled as a PPT Turing machine (see [121]) that controls all communication between parties through the following queries:

1. send(s, m) models the adversary sending a message m to session s. $\mathcal{A}$ is given the response generated by the session according to the protocol. The variables s_{sent} and s_{recv} are updated correspondingly. $\mathcal{A}$ is allowed to activate an initiator session with a peer $\hat{P}$, through send$(s, \hat{P})$ and a responder session by sending a message m to session s on behalf of $\hat{P}$ through send$(s, \hat{P}, m)$. In these cases s_{peer} is set to $\hat{P}$ and s_{role} to $\mathcal{I}$ or $\mathcal{R}$, respectively. The adversary is given the session's response according to the protocol and the variables s_{sent} and s_{recv} are initialized correspondingly.
2. corrupt$(\hat{P})$ reveals the long-term keys of party $\hat{P}$.
3. ephemeral-key(s) reveals the ephemeral secret keys of session s
4. session-key(s) reveals the session key for a completed session s.
5. test-session(s) picks a uniformly random bit b. If $b = 0$ the session-key established in session s is returned. Otherwise, a random key is returned according to the probability distribution of keys generated by the protocol. This query can only be issued to a completed session.

Definition 3.30 (**Origin sessions**, based on [4][68])**.** A possibly incomplete session s' is an origin-session for a completed session s when $s'_{sent} = s_{recv}$

Definition 3.31 (**Matching sessions**, based on [4][68])**.** Two completed sessions s and s' are matching if all of the following conditions are true:

$$s_{actor} = s'_{peer} \tag{16}$$
$$s_{peer} = s'_{actor} \tag{17}$$
$$s_{sent} = s'_{recv} \tag{18}$$
$$s_{recv} = s'_{sent} \tag{19}$$
$$s_{role} \neq s'_{role} \tag{20}$$

Definition 3.32 (**Fresh session**, based on [68])**.** A completed session s in an attack game G is fresh if all of the following conditions are true:

1. G does not include the query session-key(s).
2. For all sessions s^* such that s^* matches s, G does not include session-key(s^*).
3. G does not include both corrupt(s_{actor}) and ephemeral-key(s).
4. For all sessions s' such that s' is an origin-session for sessions s, G does not include both corrupt(s_{peer}) and ephemeral-key(s').
5. If there exists no origin-session for session s, then G does not include a corrupt(s_{peer}) query before the completion of session s.

Definition 3.33 (**Attack gamme**, based on [4], [68])**.** Security of a key-exchange protocol Π is defined via an attack game G played by an adversary $\mathcal{A}$, modeled as a PPT algorithm, against a challenger. Before the experiment starts, each party $\hat{P}$ runs a key-generation algorithm that takes a security parameter 1^λ as input and outputs the long-term keys, which may include both symmetric keys and public/private key pairs. The public keys of each party are distributed in an authenticated way to all other parties. As an amendment to the description of eCK-PFS, this thesis assumes that if the protocol requires pre-shared keys, they are distributed to the relevant parties through a secure channel. The adversary $\mathcal{A}$ is given access to all public data. The settings of the security experiment G can be described in four successive stages as follows:

1. $\mathcal{A}$ can perform send, corrupt, ephemeral-key, and session-key queries.
2. At some point in the experiment, $\mathcal{A}$ issues a test-session query to a completed session that is fresh at the time. The challenger chooses a uniformly random bit b and provides $\mathcal{A}$ with either the real session-key of the test session, if $b = 0$, or a random key from the key space, if $b = 1$.
3. $\mathcal{A}$ may continue to issue send, corrupt, ephemeral-key, and session-key queries, under the condition that the test session must remain fresh.
4. $\mathcal{A}$ outputs a bit b' as their guess for b.

$\mathcal{A}$ wins the attack game G if they correctly guess the bit b chosen by the challenger during the test-session query. Success of $\mathcal{A}$ in the experiment is expressed in terms of $\mathcal{A}$'s advantage in distinguishing whether they received the real session-key in response to the test-session query. The advantage of $\mathcal{A}$ in G against a key exchange protocol Π for security parameter λ is defined as:

$$\mathrm{Adv}_G^\Pi(\lambda) = |2P(b = b') - 1| \tag{21}$$

where $P(b = b')$ is the probability of $\mathcal{A}$ guessing correctly.

Definition 3.34 (**Negligible function** [5, p. 28])**.** A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is negligible if and only if for all $c > 0$:

$$\lim_{n \to \infty} f(n)n^c = 0 \tag{22}$$

Definition 3.35 (**Secure in eCK-PFS**, based on [4], [68])**.** A key exchange protocol Π is said to be secure in eCK-PFS if, for all PPT adversaries $\mathcal{A}$, the following conditions hold:

1. If two parties successfully complete matching sessions, then they compute the same session key.
2. $\mathcal{A}$ has no more than a negligible advantage in winning the attack game G in eCK-PFS, that is, there exists a negligible function $\text{negl}(\lambda)$ (see [5, p. 28]) in the security parameter λ such that $\text{Adv}_G^{\Pi}(\lambda) \leq \text{negl}(\lambda)$.

3.3. KNXnet/IP Secure in ISO 22510:2019

ISO 22510:2019 is the international standard that specifies the KNXnet/IP Secure protocol, which claims to provide secure communication for both unicast and multicast traffic in KNXnet/IP [6, p. 101]. It's stated goals are to ensure confidentiality, data integrity, mutual authentication and resistance against replay attacks [6, pp. 101-102]. In the protocol stack, KNXnet/IP Secure is added as a layer between KNXnet/IP and the transport layer [6, p. 103]. The protocol is contributing access control for KNXnet/IP services [6, pp. 109-110]. This permits limiting capabilities, such as the configuration a KNXnet/IP Secure devices, to specific authorized users. Furthermore, communication of the KNXnet/IP services can be encapsulated by KNXnet/IP Secure to provide encryption and authentication [6, pp. 104-105]. Supporting KNXnet/IP Secure is optional for routers and devices, but mandatory for tools like the ETS [6, p. 8].

This section provides an overview of how KNXnet/IP Secure is supposed to work according to ISO 22510:2019, in preparation of the analysis in chapter "Analysis of KNXnet/IP Secure in ISO 22510:2019". The specification can be split into two parts, unicast and multicast communication, as different approaches were used to address the stated security goals. Hence, they are explained in different subsections. However, both of them have in common that they use AES-128 with the CCM cipher mode for symmetric encryption [6, p. 102]. This is why the KNX-specific adaptation of the cipher mode is introduced first.

3.3.1. CCM

KNXnet/IP Secure does not use the canonical formatting and counter generation functions from NIST SP 800-38C, Appendix A (see [3, p. 12]) [6, pp. 107-108, pp. 119-120, pp. 127-128, pp. 130-131]. While this is permitted by the NIST standard, it requires the custom functions to fulfill the requirements from definition 3.3 in order for CCM to provide the same security properties. ISO 22510:2019 also does not directly specify a formatting and counter generation function that applies to all frames. Instead, every frame format that uses CCM specifies its B_0 and counter blocks Ctr_i. While this technically defines the counter generation function, it does not explicitly define the nonce N for both functions. More importantly, the blocks beyond B_0 are not defined in ISO 22510:2019, hence the specification is incomplete. The missing information can be found in the KNX Association's own standardization document for KNXnet/IP Secure, Application Note 159/13 v06[1], which includes a reference to the "Annex A" of the Data Secure standard for the CCM implementation [49, p. 12]. The Data Secure specification[1] defines the remaining blocks, with the notation adapted to this thesis, as [122, p. 126]:

$$B_1, ..., B_n \leftarrow \mathrm{PAD}_{16}(a \| A \| P) \tag{23}$$

Where a is a 16 bit value that contains the length of the associated data A measured in byte [122, p. 126]. While ISO 22510:2019 does not provide a formal definition of the nonce N, formatting function β and counter generation function π, it is possible to derive unambiguous ones that are consistent with the specification. Since for each respective frame format in KNXnet/IP Secure the B_0 and Ctr_i are structured so that $\mathrm{MSB}_{112}(B_0) = \mathrm{MSB}_{112}(Ctr_i)$, the nonce N is implied to be defined as $N = \mathrm{MSB}_{112}(B_0)$. Therefore, the specification implicitly defines β and π as shown in figures 11 and 12, respectively. The descriptions assume that N is supposed to be 112 bit, Q is 16 bit and i is 8 bit long. The specifications constructs B_0 to be different from the Ctr_i. Hence, the length of the plaintext P is limited by the length of i to $2^8 - 1 \cdot 16$ byte $= 4080$ byte, where 16 byte is the block size [6, p. 107]. Similarly, A may not exceed $2^{16} - 1$ byte $= 65535$ byte due to the size of a. CCM is otherwise applied as specified in NIST SP 800-38C and previously shown in figures 5 and 6.

3.3.2. Unicast

KNXnet/IP Secure specifies unicast connections that allow encrypted, authenticated and access controlled communication with the existing KNXnet/IP services. The concept of the unicast connections can be roughly summarized as follows. A client establishes a TCP connection with a KNXnet/IP Secure server [6, p. 122, p. 126]. They perform an AKE to derive a session key [6, p. 121, pp. 144-145]. During the AKE the server authenticates itself to the client by using a password-based pre-shared secret as the key [6, p. 127, p. 135]. Similarly, the client authenticates itself to the server with a password-based pre-shared secret, which is associated with a particular user on the server-side [6, pp. 129-130, p. 135]. If the authentication is successful, the server can determine what permissions the user is granted, with regard to the KNXnet/IP services it may access [6, pp. 109-110]. With the AKE completed both parties have established a KNXnet/IP Secure session and derived the associated session key, which is then used to encrypt and authenticate the communication for the duration of the session [6, p. 121, p. 144]. Within the session the client may establish any number of

[1] These documents are not available to the general public and were provided by the KNX Association

```
$\beta(N, P, A)$
1 : if $\text{len}(N) \neq 112$ then
2 :     return *Incorrect nonce length*
3 : if $\text{len}(P) > 4080 \cdot 8$ then
4 :     return *Incorrect plaintext length*
5 : if $\text{len}(A) > 65535 \cdot 8$ then
6 :     return *Incorrect associated data length*
7 : $Q \leftarrow \lceil \text{len}(P)/8 \rceil$
8 : $a \leftarrow \lceil \text{len}(A)/8 \rceil$
9 : $B_0 \leftarrow N \| Q$
10 : $B_1, ..., B_n \leftarrow \text{PAD}_{16}(a \| A \| P)$
11 : return $B_0, B_1, ..., B_n$
```

Figure 11: Formatting function, based on interpretation of ISO 22510:2019
For reference, see [6, pp. 107-108, pp. 119-120, pp. 127-128, pp. 130-131], [122, p. 126]

```
$\pi(N, m)$
1 : if $\text{len}(N) \neq 112$ then
2 :     return *Incorrect nonce length*
3 : if $m >$ 0xff then
4 :     return *Counter limit exceeded*
5 : for $i \leftarrow 0$ to $m$ do
6 :     $Ctr_i \leftarrow N \| \texttt{0xff} \| i$
7 : return $Ctr_0, Ctr_1, ..., Ctr_m$
```

Figure 12: Counter generation function, based on interpretation of ISO 22510:2019
For reference, see [6, pp. 107-108, pp. 119-120, pp. 127-128, pp. 130-131]

KNXnet/IP connections [6, p. 145]. It is also permitted to send KNXnet/IP requests that were previously considered connectionless through the session [6, p. 145]. For incoming requests the server has to ensure that the configured rules for access control are obeyed [6, pp. 109-110]. Namely, the client cannot use KNXnet/IP services without proper authorization and requests sent outside the secure session, with the insecure KNXnet/IP, might not be permitted. The discovery services from the core service family do not require the use of KNXnet/IP Secure [6, p. 136].

A more detailed explanation of unicast communication in KNXnet/IP Secure is given in the following sections.

3.3.2.1. Security Goals

The ISO 22510:2019 standard claims that the password-based authentication scheme provides mutual authentication [6, p. 121, p. 144]. Furthermore, it is supposed to be resistant against both online and offline dictionary attacks [6, pp. 121-122, p. 144]. Resistance against replay attacks is claimed to be provided by the use of sequence numbers [6, p. 122].

3.3.2.2. Configuration

The configuration of KNXnet/IP Secure is performed through interface objects and their properties, like for KNXnet/IP and KNX in general. The relevant property identifiers (PIDs) that reference the state held by KNXnet/IP servers for unicast communication are the following ones.

The `PID_DEVICE_AUTHENTICATION_CODE` contains the key that the KNXnet/IP server uses to authenticate itself to clients [6, p. 127]. The ETS generates it from a password p using PBKDF2(p, "device-authentication-code.1.secure.ip.knx.org", 65536, 128) with HMAC-SHA-256 as PRF, before writing to this property during configuration [6, p. 135]. The default value of the device authentication code is the factory default setup key (FDSK), which is printed on the label of the device [6, p. 15]. Clients need to either know the password or the resulting key for each server they communicate with in order to authenticate them during the unicast AKE. Therefore, a device authentication code is a pre-shared secret between clients and a specific server.

The `PID_PASSWORD_HASHES` is an array of password-derived keys stored by the KNXnet/IP server with which KNXnet/IP Secure clients can authenticate themselves to it [6, p. 135]. The ETS generates each of them from a respective password p using PBKDF2(p, "user-password.1.secure.ip.knx.org", 65536, 128) with HMAC-SHA-256 as PRF, before writing to this property during configuration [6, p. 135]. The default value for each entry is the key derived from the empty password [6, p. 135]. The index of the password hash in the array is the user ID, with the first entry being for the management client [6, p. 135]. This is the user with the highest permissions, which will be later explained in the "Access Control" section. There can be up to 127 passwords [6, p. 135]. Clients need to know one of the passwords and the associated user ID for their authentication to a KNXnet/IP Server during the unicast AKE. Therefore, the password hashes are a pre-shared secret between clients and a specific server.

The `PID_SECURED_SERVICE_FAMILIES` controls whether KNXnet/IP services provided by the KNXnet/IP Secure server may only be accessed through KNXnet/IP Secure or if plain KNXnet/IP may be used [6, p. 137]. This is separately configurable for the KNXnet/IP device management and configuration, routing, and tunneling services [6, p. 137]. Consequences of secure communication being required are explained in the "Access Control" section.

The `PID_TUNNELLING_USERS` stores a mapping from user IDs to indices of entries from the `PID_TUNNELLING_ADDRESSES`, which contains a subset of IAs from the additional individual addresses that the KNXnet/IP Server may use for tunneling connections [6, p. 142, p. 58]. Entries signify that the user with the respective used ID may be given access to the associated IA for tunneling connections when the use of KNXnet/IP Secure is required [6, p. 142]. The management user with ID 1 is implicitly granted access to all tunneling IAs and is not listed in the tunneling users [6, p. 142].

The ISO 22510:2019 standard only specifies properties for the server. While the KNXnet/IP Secure clients need to store device authentication codes and password hashes as well, no properties or data structures are defined for them.

3.3.2.3. Frame Formats

Unicast communication requires a handshake to establish the session key and authenticate the communication partners. As such, specific frames are required to conduct the AKE and perform further communication in the session. This section introduces the frame formats and the following sections about the AKE, finite state machine and access control explain their applications as well as the meaning of particular fields in the frames.

Definition 3.36 (**KNXnet/IP Secure header** [6, pp. 103-104])**.**
The KNXnet/IP Secure frames share a header with KNXnet/IP. Their structure is identical, but KNXnet/IP Secure introduces new service type identifiers for its frames.

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Header Length (1 byte)	Protocol Version (1 byte)
Service Type Identifier (2 byte)	
Total Length (2 byte)	

Figure 13: Structure of KNXnet/IP Secure header [6, pp. 103-104]

Definition 3.37 (`SESSION_REQUEST` [6, pp. 125-126])**.**
The `SESSION_REQUEST` frame is not wrapped in a `SECURE_WRAPPER`. It is sent without any encryption or authentication.

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Header Length (`0x06`)	Protocol Version (`0x10`)
Service Type Identifier (`0x0951`)	
Total Length (`0x26+sizeof(HPAI)`)	
HPAI Control Endpoint (variable) ⋮	
ECDH Client Public Value X (32 byte) ⋮	

Figure 14: Structure of `SESSION_REQUEST` frames [6, pp. 125-126]

Definition 3.38 (`SESSION_RESPONSE` [6, pp. 126-127])**.**
The `SESSION_RESPONSE` frame is not wrapped in a `SECURE_WRAPPER`. The server authenticates the values in A by using its device authentication code as key. The result

of CCM is stored in the “message authentication code” field.

$$A \leftarrow \text{KNXnet/IP Secure Header} \parallel \text{Secure Session Identifier} \parallel \text{X} \oplus \text{Y} \tag{24}$$

$$P \leftarrow \text{Empty} \tag{25}$$

$$B_0 \leftarrow \underbrace{\texttt{0x00} \parallel \ldots \parallel \texttt{0x00}}_{\text{16 byte}} \tag{26}$$

$$Ctr_0 \leftarrow \underbrace{\texttt{0x00} \parallel \ldots \parallel \texttt{0x00}}_{\text{14 byte}} \parallel \texttt{0xff} \parallel \texttt{0x00} \tag{27}$$

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Header Length (`0x06`)	Protocol Version (`0x10`)
Service Type Identifier (`0x0952`)	
Total Length (`0x0038`)	
Secure Session Identifier (2 byte)	
ECDH Client Public Value Y (32 byte) ⋮	
Message Authentication Code (16 byte) ⋮	

Figure 15: Structure of `SESSION_RESPONSE` frames [6, p. 126]

Definition 3.39 (`SESSION_AUTHENTICATE` [6, pp. 128-131])**.**
The `SESSION_AUTHENTICATE` frame has to be wrapped in a `SECURE_WRAPPER`. The client authenticates the values in A by using the password hash for the provided user ID as a key. The result of CCM is stored in the “message authentication code” field.

$$A \leftarrow \text{KNXnet/IP Secure Header} \parallel \texttt{0x00} \parallel \text{User ID} \parallel \text{X} \oplus \text{Y} \tag{28}$$

$$P \leftarrow \text{Empty} \tag{29}$$

$$B_0 \leftarrow \underbrace{\texttt{0x00} \parallel \ldots \parallel \texttt{0x00}}_{\text{16 byte}} \tag{30}$$

$$Ctr_0 \leftarrow \underbrace{\texttt{0x00} \parallel \ldots \parallel \texttt{0x00}}_{\text{14 byte}} \parallel \texttt{0xff} \parallel \texttt{0x00} \tag{31}$$

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Header Length (`0x06`)	Protocol Version (`0x10`)
Service Type Identifier (`0x0953`)	
Total Length (`0x0018`)	
Reserved (`0x00`)	User ID (1 byte)
Message Authentication Code (16 byte) ⋮	

Figure 16: Structure of `SESSION_AUTHENTICATE` frames [6, p. 129]

Definition 3.40 (**`SECURE_WRAPPER`** [6, pp. 106-108])**.**
The `SECURE_WRAPPER` uses the established session key to encrypt and authenticate its content. The result of CCM is stored in the "encapsulated KNXnet/IP frame" and "message authentication code" fields.

$$\begin{aligned} A &\leftarrow \text{KNXnet/IP Secure Header} \parallel \text{Secure Session Identifier} && (32)\\ P &\leftarrow \text{Encapsulated KNXnet/IP Frame} && (33)\\ B_0 &\leftarrow \text{Sequence Information} \parallel \text{Serial Number} \parallel \text{Message Tag} \parallel \text{Q} && (34)\\ Ctr_i &\leftarrow \text{Sequence Information} \parallel \text{Serial Number} \parallel \text{Message Tag} \parallel \texttt{0xff} \parallel \texttt{i} && (35) \end{aligned}$$

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Header Length (`0x06`)	Protocol Version (`0x10`)
Service Type Identifier (`0x0950`)	
Total Length (2 byte)	
Secure Session Identifier (2 byte)	
Sequence Information (6 byte) ⋮	
KNX Serial Number (6 byte) ⋮	
Message Tag (2 byte)	
Encapsulated KNXnet/IP Frame (variable) ⋮	
Message Authentication Code (16 byte) ⋮	

Figure 17: Structure of `SECURE_WRAPPER` frames [6, pp. 106-108]

Definition 3.41 (`SESSION_STATUS` [6, pp. 131-132]).
The `SESSION_STATUS` frame has to be wrapped in a `SECURE_WRAPPER` for the intended session.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Header Length (`0x06`)								Protocol Version (`0x10`)							
Service Type Identifier (`0x0954`)															
Total Length (`0x0008`)															
Status (1 byte)								Reserved (1 byte)							

Figure 18: Structure of `SESSION_STATUS` frames [6, p. 131]

3.3.2.4. Authenticated Key Exchange

The KNXnet/IP Secure handshake for establishing a session has to authenticate the parties and derive a session key, hence it as an AKE protocol. It consists of the following steps, which describe the process when no error conditions occur:

1. The KNXnet/IP Secure client establishes a TCP connection to a control endpoint of the KNXnet/IP Secure server [6, pp. 125-126]. Information about the server and its control endpoint were previously retrieved through device discovery, which does not require KNXnet/IP Secure [6, p. 109]. The client generates an ephemeral secret key $esk_{\hat{C}} \overset{R}{\leftarrow} \{0, 1, ..., 2^{256}-1\}$, which is its private key for the session. The public key X is computed as $X \leftarrow X25519(esk_{\hat{C}}, 9)$ [6, p. 125][42]. The client then sends a `SESSION_REQUEST` over the established TCP connection to the server. It contains the public key X and its own control endpoint is declared as "route back" HPAI [6, pp. 125-126]. This HPAI has the IP address and port set to zero, which indicates that the server shall use the existing TCP connection to reply [6, p. 46]. No other HPAI options are permitted to be used by ISO 22510:2019 [6, p. 126].
2. The KNXnet/IP Secure server receives the client's `SESSION_REQUEST` and verifies that the frame has a valid format. It reserves a secure session identifier, which is an ID for the session [6, p. 122, p. 144]. An ephemeral secret key $esk_{\hat{S}} \overset{R}{\leftarrow} \{0, 1, ..., 2^{256} - 1\}$ is generated, which is the server's private key for the session. Its public key Y is computed as $Y \leftarrow X25519(esk_{\hat{S}}, 9)$ [6, p. 126][42]. The server derives the shared secret $s \leftarrow \text{X25519}(esk_{\hat{S}}, X)$ [6, p. 121], as previously explained in figure 8. The session key k_s is determined by $k_s \leftarrow \text{MSB}_{128}(\text{SHA-256}(s))$ [6, p. 121]. A `SESSION_RESPONSE` is prepared, containing the secure session identifier and the server's public key Y [6, p. 126]. CCM is applied to authenticate the frame's content using the server's device authentication code k_{dac} from `PID_DEVICE_AUTHENTICATION_CODE` [6, p. 127, pp. 134-135]. The arguments are [6, p. 127-128]:
   - $T_{len} \leftarrow 128$

- $A_1 \leftarrow$ KNXnet/IP Secure Header || Secure Session Identifier || X $\oplus$ Y
- $P_1 \leftarrow$ Empty
- $N_1 \leftarrow \underbrace{\texttt{0x00}\|...\|\texttt{0x00}}_{\text{14 byte}}$

The content of the message authentication code field MAC_1 is calculated as $MAC_1 \leftarrow \text{CCM-ENC}(k_{dac}, \beta, \pi, T_{len}, N_1, P_1, A_1)$. The server sends the `SESSION_-RESPONSE` to the client over the TCP connection.

3. The client receives the `SESSION_RESPONSE` and validates the frame size [6, p. 128]. It may verify the MAC, but is not required to [6, p. 128]. If the client has knowledge of the device authentication code k_{dac}, it can construct the arguments for CCM like the server, as the KNXnet/IP Secure header, secure session identifier and the server's public key Y are contained in the `SESSION_RESPONSE` and X is the client's public key.

   - $T_{len} \leftarrow 128$
   - $A_1 \leftarrow$ KNXnet/IP Secure Header || Secure Session Identifier || X $\oplus$ Y
   - $C_1 \leftarrow$ Message Authentication Code
   - $N_1 \leftarrow \underbrace{\texttt{0x00}\|...\|\texttt{0x00}}_{\text{14 byte}}$

   The MAC can then be verified through $\text{CCM-DEC}(k_{dac}, \beta, \pi, T_{len}, N_1, C_1, A_1)$. If the MAC is valid or the validation has been skipped, the shared secret s is determined by $s \leftarrow X25519(esk_{\hat{C}}, Y)$ [6, p. 121]. The session key k_s is given by $k_s \leftarrow \text{MSB}_{128}(\text{SHA-256}(s))$ [6, p. 121].
   In order to authenticate itself to the server, the client prepares a `SESSION_-AUTHENTICATE` frame. This requires the choice of a user ID and matching password hash that corresponds with the content of the server's `PID_PASSWORD_-HASHES` [6, p. 129]. The `SESSION_AUTHENTICATE` contains the chosen user ID [6, p. 129]. CCM is applied to authenticate the frame's content using the password hash k_{pwd}. The arguments are [6, pp. 130-131]:

   - $T_{len} \leftarrow 128$
   - $A_2 \leftarrow$ KNXnet/IP Secure Header || `0x00` || User ID || X $\oplus$ Y
   - $P_2 \leftarrow$ Empty
   - $N_2 \leftarrow \underbrace{\texttt{0x00}\|...\|\texttt{0x00}}_{\text{14 byte}}$

   The content of the message authentication code field MAC_2 is calculated as $MAC_2 \leftarrow \text{CCM-ENC}(k_{pwd}, \beta, \pi, T_{len}, N_2, P_2, A_2)$.
   The `SESSION_AUTHENTICATE` frame has to be encapsulated in a `SECURE_WRAPPER` [6, p. 129]. The secure session identifier is assigned the value received in the server's `SESSION_RESPONSE` [6, p. 106], the sequence information/number starts at zero [6, p. 122], the KNX serial number for the client device is inserted and

the message tag is set to `0x0000` [6, p. 106]. CCM is applied to encrypt and authenticate the frame's content using the session key k_s. The arguments are [6, p. 107]:

- $T_{len} \leftarrow 128$
- $A_3 \leftarrow$ KNXnet/IP Secure Header $\|$ Secure Session Identifier
- $P_3 \leftarrow$ `SESSION_AUTHENTICATE` Frame
- $N_3 \leftarrow$ Sequence Information $\|$ KNX Serial Number $\|$ Message Tag

The content of the "encapsulated KNXnet/IP frame" and the MAC fields in the `SECURE_WRAPPER` are the result of CCM-ENC$(k_s, \beta, \pi, T_{len}, N_3, P_3, A_3)$, where the ciphertext is logically split into the part that is the encrypted and authenticated `SESSION_AUTHENTICATE` frame and the encrypted CBC-MAC. This `SECURE_WRAPPER` is sent to the server.

4. The server receives the `SECURE_WRAPPER` and verifies that the frame has at least the minimum required length [6, p. 108]. Furthermore, the secure session identifier has to match the current session [6, p. 108]. It decrypts the content and verifies the MAC with the session key k_s [6, p. 108]. The arguments are:

- $T_{len} \leftarrow 128$
- $A_3 \leftarrow$ KNXnet/IP Secure Header $\|$ Secure Session Identifier
- $C_3 \leftarrow$ Encapsulated KNXnet/IP frame $\|$ Message Authentication Code
- $N_3 \leftarrow$ Sequence Information $\|$ KNX Serial Number $\|$ Message Tag

The encapsulated `SESSION_AUTHENTICATE` frame P_3 is decrypted and the MAC verified by $P_3 \leftarrow$ CCM-DEC$(k_s, \beta, \pi, T_{len}, N_3, C_3, A_3)$. The length of the resulting frame is validated [6, p. 131]. Within the `SESSION_AUTHENTICATE` is the user ID with which the client attempts to authenticate itself [6, p. 129]. The server checks that its value is within the expected range and looks up the associated password hash k_{pwd} in `PID_PASSWORD_HASHES` [6, pp. 144-145]. Validation of the `SESSION_AUTHENTICATE`'s MAC is conducted by decrypting it with the k_{pwd} [6, p. 128]. The arguments are:

- $T_{len} \leftarrow 128$
- $A_2 \leftarrow$ KNXnet/IP Secure Header $\|$ `0x00` $\|$ User ID $\|$ X $\oplus$ Y
- $C_2 \leftarrow$ Message Authentication Code
- $N_2 \leftarrow \underbrace{\texttt{0x00}\|\ldots\|\texttt{0x00}}_{\text{14 byte}}$

The MAC is verified through CCM-DEC$(k_{pwd}, \beta, \pi, T_{len}, N_2, C_2, A_2)$. If this steps is successful, the client is considered authenticated and given authorization based on the permissions associated with the user ID [6, p. 131]. The client is notified of this success through a `SESSION_STATUS`, with status code `0x00` (`STATUS_AUTHENTICATION_SUCCESS`), encapsulated in a `SECURE_WRAPPER` [6, p. 145]. The server inserts its own sequence information and serial number, while the message tag is set to `0x0000`. The arguments are [6, p. 107]:

- $T_{len} \leftarrow 128$
- $A_4 \leftarrow$ KNXnet/IP Secure Header || Secure Session Identifier
- $P_4 \leftarrow$ `SESSION_STATUS` Frame
- $N_4 \leftarrow$ Sequence Information || KNX Serial Number || Message Tag

The content of the "encapsulated KNXnet/IP frame" and the MAC fields in the `SECURE_WRAPPER` are the result of CCM-ENC$(k_s, \beta, \pi, T_{len}, N_4, P_4, A_4)$, where the ciphertext is logically split into the part that is the encrypted and authenticated `SESSION_STATUS` frame and the encrypted CBC-MAC. This `SECURE_-WRAPPER` is sent to the client.

5. The client receives the `SECURE_WRAPPER` containing a `SESSION_STATUS`. It verifies that the frame has the minimum required length and the secure session identifier matches the current session [6, p. 131]. The client decrypts the content and verifies the MAC with the session key k_s [6, p. 108]. The arguments are:

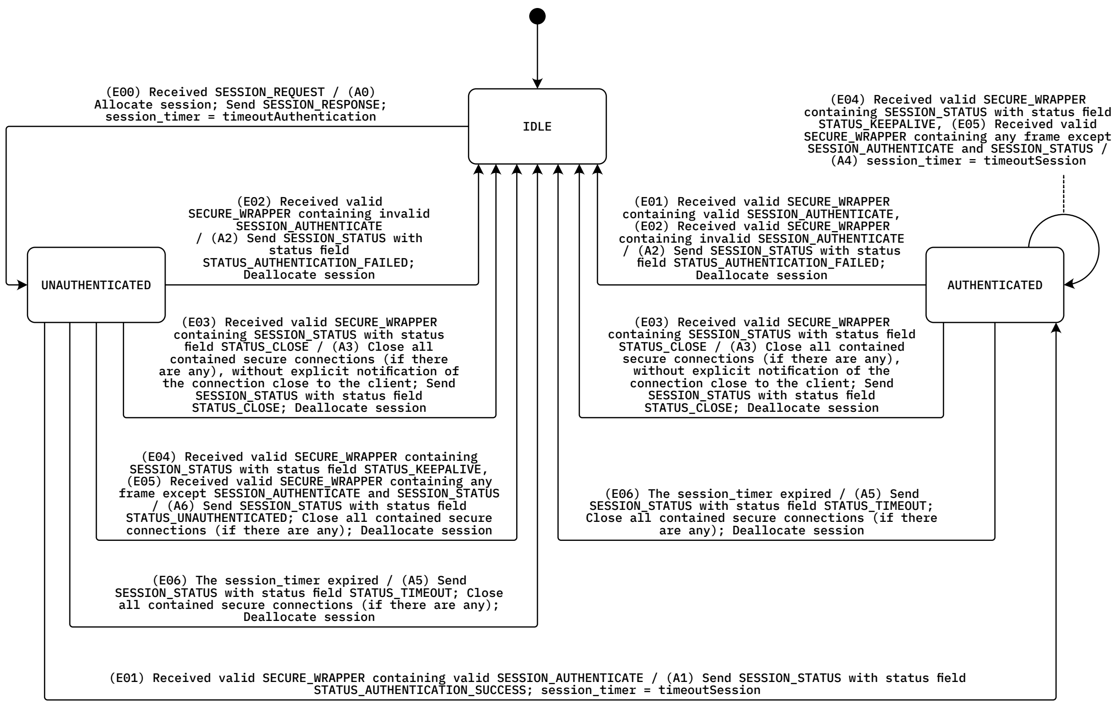
- $T_{len} \leftarrow 128$
- $A_4 \leftarrow$ KNXnet/IP Secure Header || Secure Session Identifier
- $C_4 \leftarrow$ Encapsulated KNXnet/IP frame || Message Authentication Code
- $N_4 \leftarrow$ Sequence Information || KNX Serial Number || Message Tag

The encapsulated `SESSION_STATUS` frame P_4 is decrypted and the MAC verified by $P_4 \leftarrow$ CCM-DEC$(k_s, \beta, \pi, T_{len}, N_4, C_4, A_4)$. The status code in the `SESSION_STATUS` frame informs the client about the success and the handshake is completed.

Further communication in the established session uses KNXnet/IP frames wrapped in `SECURE_WRAPPER`, which are encrypted and authenticated with the session key. Each party has to include their current sequence number in every frame and increment it by one after a frame being sent [6, p. 122]. They also have to store the last valid sequence number that they have received in the session and discard incoming frames that do not have a higher sequence number [6, p. 122].

3.3.2.5. Session Finite State Machine

ISO 22510:2019 specifies the unicast communication both textually and as a finite state machine (FSM) for the KNXnet/IP server. The tabular notation from the standard is visualized in figure 19. It describes the states and behavior of the server in the context of a session [6, p. 123]. This depiction also includes the required error handling and termination of the session, which was not included in the happy path description of the AKE in the previous section. If a session is closed, then all KNXnet/IP connections within that session are terminated as well [6, p. 145]. There is no FSM specification for the KNXnet/IP clients in ISO 22510:2019.



IDLE
UNAUTHENTICATED
AUTHENTICATED
(E00) Received SESSION_REQUEST / (A0) Allocate session; Send SESSION_RESPONSE; session_timer = timeoutAuthentication
(E02) Received valid SECURE_WRAPPER containing invalid SESSION_AUTHENTICATE / (A2) Send SESSION_STATUS with status field STATUS_AUTHENTICATION_FAILED; Deallocate session
(E03) Received valid SECURE_WRAPPER containing SESSION_STATUS with status field STATUS_CLOSE / (A3) Close all contained secure connections (if there are any), without explicit notification of the connection close to the client; Send SESSION_STATUS with status field STATUS_CLOSE; Deallocate session
(E04) Received valid SECURE_WRAPPER containing SESSION_STATUS with status field STATUS_KEEPALIVE, (E05) Received valid SECURE_WRAPPER containing any frame except SESSION_AUTHENTICATE and SESSION_STATUS / (A6) Send SESSION_STATUS with status field STATUS_UNAUTHENTICATED; Close all contained secure connections (if there are any); Deallocate session
(E06) The session_timer expired / (A5) Send SESSION_STATUS with status field STATUS_TIMEOUT; Close all contained secure connections (if there are any); Deallocate session
(E01) Received valid SECURE_WRAPPER containing valid SESSION_AUTHENTICATE / (A1) Send SESSION_STATUS with status field STATUS_AUTHENTICATION_SUCCESS; session_timer = timeoutSession
(E04) Received valid SECURE_WRAPPER containing SESSION_STATUS with status field STATUS_KEEPALIVE, (E05) Received valid SECURE_WRAPPER containing any frame except SESSION_AUTHENTICATE and SESSION_STATUS / (A4) session_timer = timeoutSession
(E01) Received valid SECURE_WRAPPER containing valid SESSION_AUTHENTICATE, (E02) Received valid SECURE_WRAPPER containing invalid SESSION_AUTHENTICATE / (A2) Send SESSION_STATUS with status field STATUS_AUTHENTICATION_FAILED; Deallocate session
(E03) Received valid SECURE_WRAPPER containing SESSION_STATUS with status field STATUS_CLOSE / (A3) Close all contained secure connections (if there are any), without explicit notification of the connection close to the client; Send SESSION_STATUS with status field STATUS_CLOSE; Deallocate session
(E06) The session_timer expired / (A5) Send SESSION_STATUS with status field STATUS_TIMEOUT; Close all contained secure connections (if there are any); Deallocate session

Figure 19: Session state machine for the server, based on [6, pp. 123-125]

3.3.2.6. Access Control

During the AKE the client authenticates itself with a user ID and password hash to the server. This information is also used for access control, which determines the services that the client may access, if the authentication is successful [6, p. 109]. The user ID `0x01` is for management level access and `0x02` to `0x7f` for user level access [6, p. 129]. As previously explained, the `PID_SECURED_SERVICE_FAMILIES` configures whether it is required to use KNXnet/IP Secure for the connection-oriented KNXnet/IP services device management and tunneling. When KNXnet/IP Secure is not required, then both service can be access through plain KNXnet/IP connections, without any encryption and authentication [6, pp. 109-110]. Clients may also create KNXnet/IP Secure sessions, in which case all user IDs are permitted to establish device management and tunneling connections within the session [6, pp. 109-110]. However, if `PID_SECURED_SERVICE_FAMILIES` enforces KNXnet/IP Secure for device management connections, then plain KNXnet/IP is not permitted and only the user ID `0x01` may create device management connections in KNXnet/IP Secure sessions [6, pp. 109-110]. When `PID_SECURED_SERVICE_FAMILIES` requires KNXnet/IP Secure for tunneling connections, then the user ID `0x01` may pick any IA from `PID_TUNNELLING_ADDRESSES` for its tunnel [6, p. 142]. Users `0x02` to `0x7f` may only request an IA for which they have an entry in `PID_TUNNELLING_USERS` [6, p. 143]. If they do not request a specific IA the server still has to check if an IA is available that given user ID is allowed to use [6, p. 143]. The discovery services are not access restricted [6, p. 109].

3.3.3. Multicast

KNXnet/IP Secure specifies encrypted and authenticated multicast communication. The concept can be roughly summarized as follows. Due to KNXnet/IP, all routing-capable KNXnet/IP Secure devices are part of a multicast group that they have joined through IGMP [6, p. 82, p. 111]. They can communicate via UDP by sending their frames to port 3671 of the group's multicast address [6, p. 82]. The definition of group membership for KNXnet/IP Secure also requires knowledge of the backbone key [6, p. 111]. Therefore, it is a pre-shared secret. This key is used to encrypt and authenticate all multicast communication of the group [6, p. 134]. It has an unlimited lifetime [6, p. 111], thus it is not updated after the initial configuration. The KNXnet/IP Secure protocol specifies a procedure to synchronize timers among the group members [6, p. 112]. The timer is used to reject old frames [6, p. 111], similar to the sequence numbers in unicast communication.

3.3.3.1. Security Goals

Besides the general security goals for KNXnet/IP Secure, the specification claims that synchronized timers would provide a defense against replay attacks [6, p. 111]. At the same time ISO 22510:2019 also states that frames with "slightly past timer values" [6, p. 111] will still be accepted. Multicast communication is supposed to provide mutual authentication [6, p. 102].

3.3.3.2. Configuration

The previously introduced backbone key is configured through `PID_BACKBONE_KEY` [6, p. 134]. During the setup of the project the ETS has to generate a backbone key for each multicast group [6, p. 134]. It is set for the relevant KNXnet/IP Secure devices during their initial configuration.

The `PID_SECURED_SERVICE_FAMILIES` controls whether the routing service family is allowed to be accessed through KNXnet/IP or if KNXnet/IP Secure is required [6, p. 109, p. 137]. Unlike unicast access to connection-oriented services, the use of KNXnet/IP Secure for multicast communication is only allowed when secure communication is required [6, p. 110]. Details are provided in the "Access Control" section.

There are two configurable properties for the timer synchronization. The `PID_-MULTICAST_LATENCY_TOLERANCE` determines how far in the past the received timer in a frame is allowed to be in relation to the recipient's multicast timer, during which the frame will still be accepted [6, p. 113, p. 141]. The selection of this value is a trade-off between handling network latencies and avoiding replay attacks [6, p. 141]. The default latency tolerance is 2 seconds [6, p. 141]. `PID_SYNC_LATENCY_FRACTION` is the fraction of the latency tolerance that determines how far in the past or future the received timers may be, relative to the recipient's own timer, in order the cancel the scheduled sending of a `TIMER_NOTIFY` to the group [6, p. 115, p. 141]. The default value is 10.2% [6, p. 142]. Its value is supposed to represent the common case of latency [6, p. 113]. Details are provided in the "Timer Synchronization" section.

3.3.3.3. Frame Formats

Multicast communication requires frames for the synchronization of timers and communication with the KNXnet/IP routing service. This section introduces the frame formats and the following sections about the communication, finite state machine and access control explain their applications as well as the meaning of particular fields.

Definition 3.42 (**`SECURE_WRAPPER`** [6, pp. 106-108]).
The `SECURE_WRAPPER` frame is encrypted and authenticated with the backbone key. Its structure is the same as in unicast, but the fields are used for a different purpose. The secure session identifier is set to a constant `0x0000`, the sequence information contain the sender's multicast timer and the message tag is assigned a random value for each frame [6, p. 106].

$$\begin{aligned} A &\leftarrow \text{KNXnet/IP Secure Header} \parallel \text{Secure Session Identifier} && (36)\\ P &\leftarrow \text{Encapsulated KNXnet/IP Frame} && (37)\\ B_0 &\leftarrow \text{Sequence Information} \parallel \text{Serial Number} \parallel \text{Message Tag} \parallel \text{Q} && (38)\\ Ctr_i &\leftarrow \text{Sequence Information} \parallel \text{Serial Number} \parallel \text{Message Tag} \parallel \texttt{0xff} \parallel \texttt{i} && (39) \end{aligned}$$

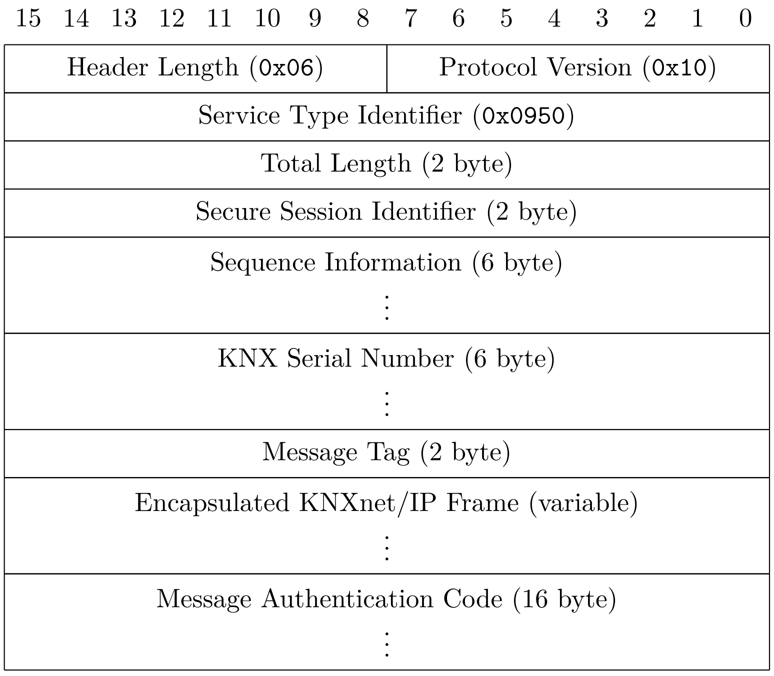

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Header Length (0x06)
Protocol Version (0x10)
Service Type Identifier (0x0950)
Total Length (2 byte)
Secure Session Identifier (2 byte)
Sequence Information (6 byte)
KNX Serial Number (6 byte)
Message Tag (2 byte)
Encapsulated KNXnet/IP Frame (variable)
Message Authentication Code (16 byte)

Figure 20: Structure of `SECURE_WRAPPER` frames [6, pp. 106-108]

Definition 3.43 (**`TIMER_NOTIFY`** [6, pp. 118-120]).
The `TIMER_NOTIFY` frame is not wrapped in a `SECURE_WRAPPER`. The values in A are

authenticated with the backbone key.

$$A \leftarrow \text{KNXnet/IP Secure Header} \tag{40}$$
$$P \leftarrow \text{Empty} \tag{41}$$
$$B_0 \leftarrow \text{Timer Value} \parallel \text{Serial Number} \parallel \text{Message Tag} \parallel \texttt{0x0000} \tag{42}$$
$$Ctr_0 \leftarrow \text{Timer Value} \parallel \text{Serial Number} \parallel \text{Message Tag} \parallel \texttt{0xff00} \tag{43}$$

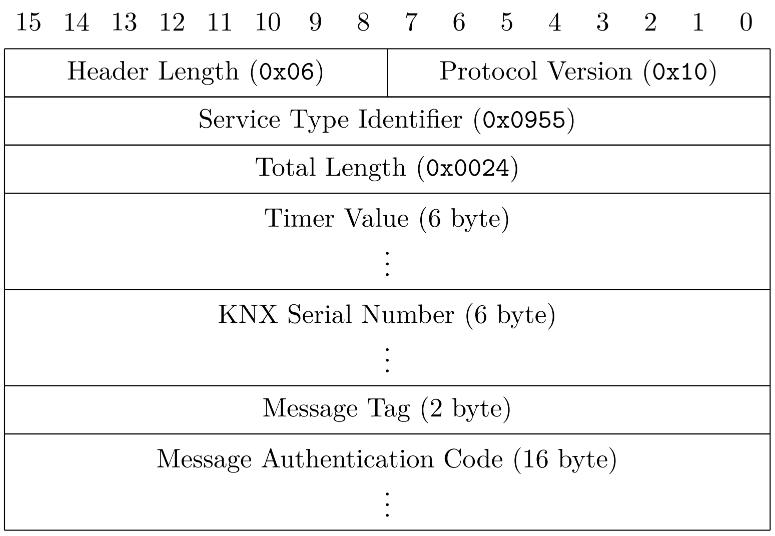


Figure 21: Structure of `TIMER_NOTIFY` frames [6, p. 118]

3.3.3.4. Communication

Since the multicast communication is connectionless each group member can send `SECURE_WRAPPER` and `TIMER_NOTIFY` to all other group members without a prior handshake. Group membership, as defined by KNXnet/IP Secure, requires knowledge of the backbone key by all members [6, p. 111]. Hence, they are able to decrypt the incoming frames. Encrypting the `SECURE_WRAPPER` with the backbone key k_b through CCM-ENC$(k_b, \beta, \pi, T_{len}, N, P, A)$ requires:

- $T_{len} \leftarrow 128$
- $A \leftarrow$ KNXnet/IP Secure Header $\parallel$ Secure Session Identifier
- $P \leftarrow$ KNXnet/IP Frame
- $N \leftarrow$ Sequence Information $\parallel$ Serial Number $\parallel$ Message Tag

Similarly the arguments for the `TIMER_NOTIFY` are:

- $T_{len} \leftarrow 128$
- $A \leftarrow$ KNXnet/IP Secure Header $\parallel$ Secure Session Identifier
- $P \leftarrow$ Empty

- $N \leftarrow$ Timer Value || Serial Number || Message Tag

The sequence information in `SECURE_WRAPPER` and timer value in `TIMER_NOTIFY` both contain the current timer value of the sender. It could happen that two parties send frames with identical timer values, hence the message tag is a random value for each frame to prevent nonce reuse [6, p. 106, p. 108, p. 119].

3.3.3.5. Timer Synchronization Finite State Machine

KNXnet/IP Secure's protection against replay attacks in multicast communication relies on timers. Each device has a 48 bit timer that it increments monotonically at least once a millisecond [6, p. 111, p. 133]. ISO 22510:2019 requires that it is stored persistently [6, p. 133]. It may not decrease, even when the device loses power [6, p. 111, p. 133]. If a device is not capable of immediately persisting the timer when a low power condition is detected, it has to store the timer in regular intervals measured in the multicast timer domain [6, p. 133]. On power-up the multicast timer then has to be incremented by the persistence interval [6, p. 133].

The idea of rejecting frames, based on the detection of outdated timers contained within, requires the synchronization of timer values among the multicast group members. The general concept is that every device receiving a valid frame with a larger timer value than its own, updates its internal timer to match the value [6, p. 111]. If frames are received with timers that are more than `PID_MULTICAST_LATENCY_TOLERANCE` behind the recipient's own timer, they are rejected [6, p. 111]. `TIMER_NOTIFY` frames are supposed to be sent periodically, or when outdated timers are received by members of the group, to synchronize the timers [6, p. 112]. Devices can take on the role of "time keeper" when they have not received frames within the expected fraction of the latency tolerance determined by `PID_SYNC_LATENCY_FRACTION`. In this case the device will send `TIMER_NOTIFY` frames more frequently until it receives a larger timer value itself [6, pp. 112-113]. Device that are not time keepers are referred to as "time followers" [6, p. 116]. This process is specified by the standard with a FSM in tabular notation which is visualized in figure 22. Its parameters are defined in table 2.

Initially the timer synchronization starts in the `SCHED_PERIODIC` state and it picks are random `notify_timer` as shown in figure 22. The purpose is to prevent a flood of messages when all devices start simultaneously [6, p. 117]. If a previous value for the multicast timer (`mc_timer`) has been persisted, then this value is restored and possibly incremented by the persistence interval [6, p. 117]. Whenever a device is assigned a new backbone key, which is semantically equivalent to joining a new multicast group, the multicast timer is reset to zero [6, p. 112]. The reference in the FSM to "section 5.7.2.2.3" refers to the behavior described in this paragraph. Since the initialization of the timer synchronization does not prevent replay attacks before the first authentic and current multicast timer is received, the device may optionally implement a procedure that supposedly allows to acquire an authentic timer [6, pp. 117-118]:

1. Set `mc_timer_authentic = false`.

2. Do not process SECURE_WRAPPER until mc_timer_authentic == true.
3. Send or schedule TIMER_NOTIFY and remember the used message tag.
4. Wait for maxDelayTimeFollowerUpdateNotify $+ 2 \times$ latencyTolerance after the first TIMER_NOTIFY or SECURE_WRAPPER is sent or received
   a) If TIMER_NOTIFY is received that contains own serial number and remembered tag, then mc_timer_authentic = true and stop waiting.
5. The most recent timer of all received TIMER_NOTIFY and SECURE_WRAPPER is assigned to mc_timer and mc_timer_authentic = true.

These steps are not included in the FSM specification. ISO 22510:2019 assumes that there is at least one other honest party that sends authentic timers [6, p. 118].

Variable	**Value**
latencyTolerance	PID_MULTICAST_LATENCY_TOLERANCE
syncLatencyTolerance	PID_SYNC_LATENCY_FRACTION applied to PID_MULTICAST_LATENCY_TOLERANCE
minDelayInitialNotify	0 s
maxDelayInitialNotify	10 s
minDelayTimeKeeperPeriodicNotify	0 s
minDelayTimeKeeperUpdateNotify	0.1 s
maxDelayTimeKeeperPeriodicNotify	minDelayTimeKeeperPeriodicNotify $+ 3 \times$ syncLatencyTolerance
minDelayTimeFollowerPeriodicNotify	maxDelayTimeKeeperPeriodicNotify + syncLatencyTolerance
maxDelayTimeFollowerPeriodicNotify	minDelayTimeFollowerPeriodicNotify $+ 10 \times$ syncLatencyTolerance
maxDelayTimeKeeperUpdateNotify	minDelayTimeKeeperUpdateNotify + syncLatencyTolerance
minDelayTimeFollowerUpdateNotify	maxDelayTimeKeeperUpdateNotify + syncLatencyTolerance
maxDelayTimeFollowerUpdateNotify	minDelayTimeFollowerUpdateNotify $+ 10 \times$ syncLatencyTolerance

Table 2: Timer synchronization parameters, based on [6, pp. 113-114]

3.3.3.6. Access Control

The PID_SECURED_SERVICE_FAMILIES allows to configure whether secure communication shall be enabled for the routing service family [6, p. 137]. When it is, no plain

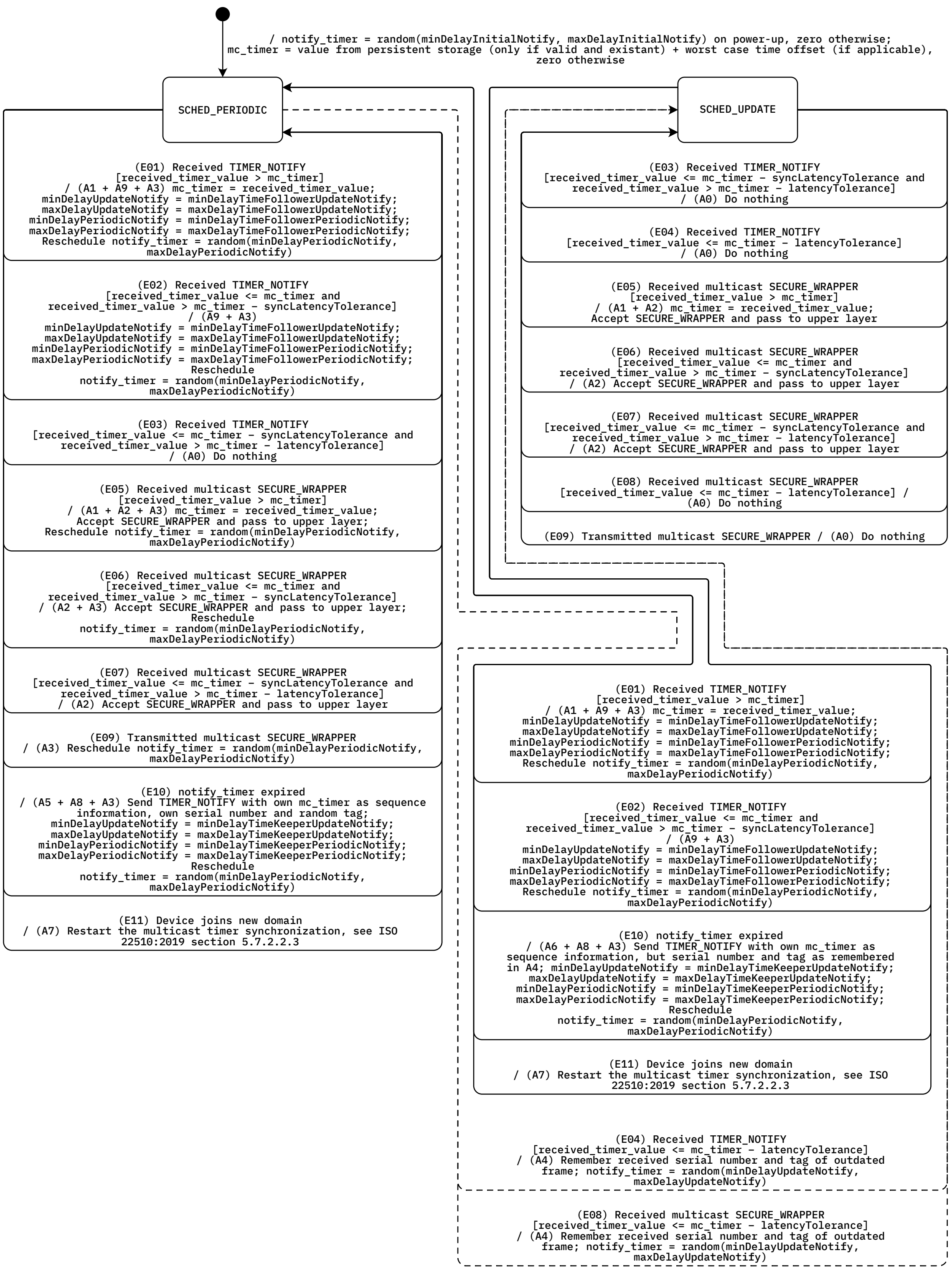

/ notify_timer = random(minDelayInitialNotify, maxDelayInitialNotify) on power-up, zero otherwise;
mc_timer = value from persistent storage (only if valid and existant) + worst case time offset (if applicable), zero otherwise
SCHED_PERIODIC
SCHED_UPDATE
(E01) Received TIMER_NOTIFY [received_timer_value > mc_timer] / (A1 + A9 + A3) mc_timer = received_timer_value; minDelayUpdateNotify = minDelayTimeFollowerUpdateNotify; maxDelayUpdateNotify = maxDelayTimeFollowerUpdateNotify; minDelayPeriodicNotify = minDelayTimeFollowerPeriodicNotify; maxDelayPeriodicNotify = maxDelayTimeFollowerPeriodicNotify; Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E02) Received TIMER_NOTIFY [received_timer_value <= mc_timer and received_timer_value > mc_timer - syncLatencyTolerance] / (A9 + A3) minDelayUpdateNotify = minDelayTimeFollowerUpdateNotify; maxDelayUpdateNotify = maxDelayTimeFollowerUpdateNotify; minDelayPeriodicNotify = minDelayTimeFollowerPeriodicNotify; maxDelayPeriodicNotify = maxDelayTimeFollowerPeriodicNotify; Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E03) Received TIMER_NOTIFY [received_timer_value <= mc_timer - syncLatencyTolerance and received_timer_value > mc_timer - latencyTolerance] / (A0) Do nothing
(E05) Received multicast SECURE_WRAPPER [received_timer_value > mc_timer] / (A1 + A2 + A3) mc_timer = received_timer_value; Accept SECURE_WRAPPER and pass to upper layer; Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E06) Received multicast SECURE_WRAPPER [received_timer_value <= mc_timer and received_timer_value > mc_timer - syncLatencyTolerance] / (A2 + A3) Accept SECURE_WRAPPER and pass to upper layer; Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E07) Received multicast SECURE_WRAPPER [received_timer_value <= mc_timer - syncLatencyTolerance and received_timer_value > mc_timer - latencyTolerance] / (A2) Accept SECURE_WRAPPER and pass to upper layer
(E09) Transmitted multicast SECURE_WRAPPER / (A3) Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E10) notify_timer expired / (A5 + A8 + A3) Send TIMER_NOTIFY with own mc_timer as sequence information, own serial number and random tag; minDelayUpdateNotify = minDelayTimeKeeperUpdateNotify; maxDelayUpdateNotify = maxDelayTimeKeeperUpdateNotify; minDelayPeriodicNotify = minDelayTimeKeeperPeriodicNotify; maxDelayPeriodicNotify = maxDelayTimeKeeperPeriodicNotify; Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E11) Device joins new domain / (A7) Restart the multicast timer synchronization, see ISO 22510:2019 section 5.7.2.2.3
(E03) Received TIMER_NOTIFY [received_timer_value <= mc_timer - syncLatencyTolerance and received_timer_value > mc_timer - latencyTolerance] / (A0) Do nothing
(E04) Received TIMER_NOTIFY [received_timer_value <= mc_timer - latencyTolerance] / (A0) Do nothing
(E05) Received multicast SECURE_WRAPPER [received_timer_value > mc_timer] / (A1 + A2) mc_timer = received_timer_value; Accept SECURE_WRAPPER and pass to upper layer
(E06) Received multicast SECURE_WRAPPER [received_timer_value <= mc_timer and received_timer_value > mc_timer - syncLatencyTolerance] / (A2) Accept SECURE_WRAPPER and pass to upper layer
(E07) Received multicast SECURE_WRAPPER [received_timer_value <= mc_timer - syncLatencyTolerance and received_timer_value > mc_timer - latencyTolerance] / (A2) Accept SECURE_WRAPPER and pass to upper layer
(E08) Received multicast SECURE_WRAPPER [received_timer_value <= mc_timer - latencyTolerance] / (A0) Do nothing
(E09) Transmitted multicast SECURE_WRAPPER / (A0) Do nothing
(E01) Received TIMER_NOTIFY [received_timer_value > mc_timer] / (A1 + A9 + A3) mc_timer = received_timer_value; minDelayUpdateNotify = minDelayTimeFollowerUpdateNotify; maxDelayUpdateNotify = maxDelayTimeFollowerUpdateNotify; minDelayPeriodicNotify = minDelayTimeFollowerPeriodicNotify; maxDelayPeriodicNotify = maxDelayTimeFollowerPeriodicNotify; Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E02) Received TIMER_NOTIFY [received_timer_value <= mc_timer and received_timer_value > mc_timer - syncLatencyTolerance] / (A9 + A3) minDelayUpdateNotify = minDelayTimeFollowerUpdateNotify; maxDelayUpdateNotify = maxDelayTimeFollowerUpdateNotify; minDelayPeriodicNotify = minDelayTimeFollowerPeriodicNotify; maxDelayPeriodicNotify = maxDelayTimeFollowerPeriodicNotify; Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E10) notify_timer expired / (A6 + A8 + A3) Send TIMER_NOTIFY with own mc_timer as sequence information, but serial number and tag as remembered in A4; minDelayUpdateNotify = minDelayTimeKeeperUpdateNotify; maxDelayUpdateNotify = maxDelayTimeKeeperUpdateNotify; minDelayPeriodicNotify = minDelayTimeKeeperPeriodicNotify; maxDelayPeriodicNotify = maxDelayTimeKeeperPeriodicNotify; Reschedule notify_timer = random(minDelayPeriodicNotify, maxDelayPeriodicNotify)
(E11) Device joins new domain / (A7) Restart the multicast timer synchronization, see ISO 22510:2019 section 5.7.2.2.3
(E04) Received TIMER_NOTIFY [received_timer_value <= mc_timer - latencyTolerance] / (A4) Remember received serial number and tag of outdated frame; notify_timer = random(minDelayUpdateNotify, maxDelayUpdateNotify)
(E08) Received multicast SECURE_WRAPPER [received_timer_value <= mc_timer - latencyTolerance] / (A4) Remember received serial number and tag of outdated frame; notify_timer = random(minDelayUpdateNotify, maxDelayUpdateNotify)

Figure 22: Timer synchronization state machine, based on [6, pp. 113-117]

KNXnet/IP frames with routing indications may be accepted by the device [6, p. 110]. It enables the timer synchronization and all routing frames being sent have to be wrapped in `SECURE_WRAPPER` [6, p. 110]. If the secure communication is disabled, then only plain KNXnet/IP frames are allowed to be sent and received [6, p. 110]. Additionally, the timer synchronization is disabled, since sending and receiving `TIMER_NOTIFY` frames is prohibited [6, p. 110].

KNXnet/IP Secure disallows access to the remote configuration and diagnosis service family when the use of Data Secure is required [6, p. 110], [122, p. 56]. The `PID_SECURED_SERVICE_FAMILIES` does not provide an entry for this service [6, p. 137]. Hence, it is not possible to restrict access for KNXnet/IP through this property. ISO 22510:2019 does not specify limitations for accessing remote configuration through KNXnet/IP when Data Secure is disabled.

3.4. Model Checking

Model checking is a technique for the formal verification of properties in a given model, such as the FSMs in the specification of KNXnet/IP Secure. It is a software-aided approach, where, given a model description and a desired property, the goal is to find a counterexample that refutes the stated property, if it is not satisfied by the model. Depending on the approach it may also be possible to infer that the property is satisfied, if it can be proven that no such counterexample exists. Model checking is useful because it allows to verify that the model fulfills the expected properties of the real system it attempts to abstract. Under the assumption that the model is correct, it is also possible to analyze properties in the model for a better understanding of the real system. Since the software requires a formal description of the model, it also aids in the identification of inconsistencies and ambiguities in the textual description of the system's specification.

As previously stated, this thesis applies the symbolic model checking software NuXMV [75] in chapter "Analysis of KNXnet/IP Secure in ISO 22510:2019" to analyze the properties of the FSMs in ISO 22510:2019 and in chapter "Analysis of Certified Devices" to identify differences between the behavior of the certified devices and the specification. This software was chosen over NuSMV and Uppaal because it permits the modeling of infinite-state systems [75]. In particular, it supports unbounded data types [75], which allow a straightforward transfer of the timer synchronization FSM specification into the modeling language. Unlike Uppaal it can model clocks with different growth rates. Furthermore, NuXMV handles the state complexity for both BDD- and BMC-based model checking reasonably well for the applications in this work.

NuXMV requires the description of the model in its input language, which is documented in the user manual [123]. A FSM can be described through state variables, their initial values and assignment constraints that define under which conditions a new state is reached. The conditions can for example contain boolean expressions, comparisons, arithmetic and bitwise operations [123, pp. 13-14]. Random values can be modeled through range constants or sets, which define the possible values a variable may take. This is required for the description of the timer synchronization FSM. Additionally, invariants can be defined that are required to hold for the model. Constraints such as timer progression having to be a positive value can be enforced like this. A simple example of the NuXMV notation is provided in figure 23. It depicts a FSM with three states that reacts to events modeled as an integer.

```
MODULE fsm(event)
  VAR
    _state_ : { S1, S2, S3 };

  ASSIGN
    init(_state_) := S1;
    next(_state_) := case
      _state_ = S1 & event = 1 : S2;
      _state_ = S2 & event = 2 : S3;
      _state_ = S3 & event = 3 : S1;
      TRUE : _state_;
    esac;

MODULE main
  VAR
    fsm : fsm(event);
    event : 1..3;
```

Figure 23: NuXMV syntax

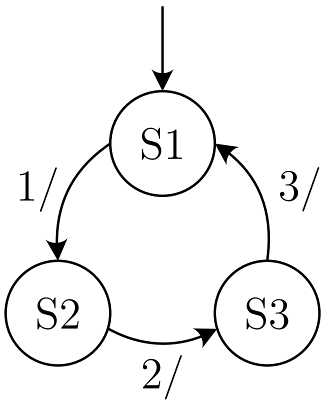


Figure 24: Finite state machine

The properties for the evaluation of the model are expressed through the temporal logics CTL [77] and LTL [76]. The former can be evaluated based on BDD [78], which can either provide a counterexample and if none can be found, proves that the property holds. The latter is analyzed through BMC [79] with a SAT- or SMT-solver, that can either provide a counterexample or prove that no counterexample for the given bound exists. It does not prove that the property holds in general. This would require the bound to be large enough to cover all possible states in the evaluation that could produce a counterexample. The research by Clarke et al. [80] describes the necessary conditions for the bound to ensure completeness for a given model. For both approaches it is required that the execution finishes in order to draw conclusions. The model complexity may cause a state explosion that exceeds available resources and thus the computation does not terminate within a reasonable time frame.

This work uses a subset of the CTL and LTL for its evaluation of FSMs. Besides the general boolean expressions, the semantics of the following NuXMV notation in CTL and LTL are relevant.

Definition 3.44 (CTL quantifier subset in NuXMV based on [123, p. 40]**).**
Let p be a CTL expression:

- "**EX** p" is true in a state s if there exists a state s' such that a transition goes from s to s' and p is true in s'.

- "**AX** p" is true in a state s if for all states s' where there is a transaction from s to s', p is true in s'.
- "**AG** p" is true in a state s_0 if for all infinite series of transitions $s_0 \to s_1, s_1 \to s_2, ..., s_{\infty-1} \to s_\infty$ such that p is true in every s_i.
- "**EF** p" is true in a state s_0 if there exists a series of transitions $s_0 \to s_1, s_1 \to s_2, ..., s_{n-1} \to s_n$ such that p is true in s_n.

Definition 3.45 (LTL quantifier subset in NuXMV based on [123, p. 40]**).**
Let p be an LTL expression:

- "**X** p" is true at time t if p is true at time $t + 1$
- "**F** p" is true at time t if p is true at some time $t' \geq t$
- "**G** p" is true at time t if p is true at all times $t' \geq t$

An example of checks that can be conducted with CTL on every FSM would be the reachability of states, activation of transitions and whether the FSM is deterministic. In the FSM shown in figure 23, the reachability check of the state `S1` could be expressed as `CTLSPEC EF _state_ = S1;` and similarly for the other states. As per the definition provided above, the property is satisfied if a series of transitions from the initial state to the target state exist. A nearly identical check can identify whether a transition is able to fire in the FSM. In the example FSM this would be `CTLSPEC EF event = 1;` for the transition between `S1` and `S2`. In more complex examples the result would be less obvious because the activation of the transition can depend on many state variables and they may take different values depending on the order of events that occur. Finally, it is also possible to check for non-determinism. This would generally require to evaluate if two transitions can be simultaneously active in the current state. All combinations of possible pairings need to be evaluated for each respective state. Practically this could be implemented by setting a variable `nondeterministic` to true in an `ASSIGN`-statement when two transitions in a state can be active at the same time. The combinations of pairings per state could be programmatically generated. This would allow to evaluate if the FSM is deterministic with `CTLSPEC AG !nondeterministic;`. When a CTL or LTL statement is not satisfied and NuXMV is able to identify a counterexample, it outputs a trace of variables and transitions that lead to a state that proves the violation of the assumption. It should be noted that by default the execution in NuXMV is deterministic, because it will always pick the first listed transition that is active. This has to be considered when non-determinism is intended.

3.5. Protocol State Fuzzing

Protocol state fuzzing is a black-box testing technique developed by de Ruiter and Poll [81] to infer the FSM of a system under test (SUT) from its observable behavior. A software that performs the protocol state fuzzing is referred to as the state learner. It requires an alphabet based on which the inference of SUT's FSM occurs. The alphabet contains identifiers which represent frames that the SUT is expected to send or receive. The state learner has to implement a mapping from elements of the alphabet to actual frames and vice versa. In general, the FSM of the SUT is learned by iteratively refining a hypothesis about its FSM through requests sent to the SUT and comparing the received reply to the expected reply of the hypothesized FSM. During the process the mapping between the alphabet and the frames is required, because the generated test queries for the SUT are based on the alphabet and have to be transformed into frames that can be sent to it. Similarly, frames received by the state learner from the SUT have to be translated back to the alphabet for the comparison with the hypothesis. If the state learner is able to find a counterexample, where the observed behavior does not match expected model, it uses the information to improve the hypothesis until either no difference can be found or configured limits are reached. Since the inference of protocol state fuzzing is based on the observable reaction of the SUT to requests, this method cannot properly learn the SUT's internal state when it depends on other factors than the requests sent by the state learner. This particularly includes timer-controlled behavior as seen in the timer synchronization FSM.

The protocol state fuzzing in this work is based on the open source implementation by de Ruiter and Poll [81], [82], hence it uses LearnLib [83] as well. Hypotheses about the SUT's FSM are generated with a version of Angluin's L^*-method [124]. Equivalence tests are conducted with a modified version of Chow's W-method [125]. The resulting counterexamples are used for the L^*-method to improve the hypothesis. Further details about the methodology are provided in the "Analysis of Certified Devices" chapter.

3.6. Risk Analysis with BSI 200-3

BSI 200-3 [85] is a standard by the Bundesamt für Sicherheit in der Informationstechnik (BSI) that specifies the methodology for risk analysis in organizations with regard to IT-security. The prerequisite is the application of BSI 200-2 [89], which means the following steps are suggested:

1. A security scope has to be chosen, which determines the extent of the assessment:
   - "Basic protection" has the objective to ensure a broad and basic initial safeguarding [89, p. 25].
   - "Core protection" has the objective to protect assets that face a higher risk [89, p. 26].
   - "Standard protection" has the objective to protect all areas in depth [89, p. 26].
2. A structure analysis has to be conducted that identifies business processes, applications, IT systems, industrial control systems, internet of things (IoT) devices, rooms and buildings, and communications links, that are being used or are in development [89, p. 65]. A network plan has to be created [89, p. 71]. It is meant to visualize the components involved and their network connections.

3. The protection requirements of every target object has to be determined with regard to confidentiality, integrity and authentication (CIA) [89, p. 78]. There are three categories which represent the consequences that are expected when a violation of one of the respective CIA-properties occurs [89, p. 79]:
   - "Normal", the effects of damage are limited and manageable.
   - "High", the effects of damage can be considerable.
   - "Very high", the effects of the damage can reach catastrophic levels that threaten the existence of the organization.
4. Every target object needs to be mapped to a module provided by the "IT-Grundschutz" compendium ([91]) [85, p. 8]. It contains threat scenarios for the respective modules and security requirements that need to be met for the different security scopes. This process is referred to as "modelling" [89, p. 94].
5. The "IT-Grundschutz check" should be performed to determine what requirements from the compendium have already been met and which issues remain insufficiently addressed [85, p. 8] [89, p. 104].

Once the preparations with BSI 200-2 are completed, the risk analysis with BSI 200-3 can be conducted:

1. A threat overview needs to be created that determines for every target object whether elementary risks listed in the "IT-Grundschutz" compendium apply to it [85, p. 13]. For target objects that map to an existing module in the compendium, it already provides a list of applicable elementary threats [85, p. 13], [91]. Additional elementary threats that may apply to a target object have to be categorized into "Directly relevant", "Indirectly relevant", and "Not relevant" [85, p. 13].
2. A risk assessment has to be performed to determine the extent of damage and frequency of occurrence for every relevant threat per target object [85, p. 21]. This could be conducted based on empirical data for a quantitative analysis or with qualitative categories. The latter is the suggested approach [85, p. 21]. The categories for the frequency are [85, p. 21]:
   - "Rarely", it is expected to occur at most every 5 years.
   - "Medium", it is expected to occur once a year to once every 5 years.
   - "Frequently", it is expected to occur once a month to once a year.
   - "Very frequently", it is expected to occur several times a month.

   The categories for the severity of the damage are [86, p. 21-22]:
   - "Negligible", the damage is low and can be ignored.
   - "Limited", the damage is limited and manageable.
   - "Considerable", the damage is considerable.
   - "Existence threatening", the damage can reach a catastrophic level that threatens the existence of the organization.

3. The risk for each threat is determined by the risk matrix depicted in figure 25. The target object does not provide adequate protection for those that are classified as “high” and “very high” [85, p. 23].
4. Treatment of the risk can be accomplished through avoidance, reduction, transfer or acceptance [85, pp. 27-28]. The choice of treatment and additional safeguards should be documented. Afterwards, the risk category has to be estimated again, with the risk treatment applied [85, p. 29].
5. Consolidation requires the evaluation of the additional security safeguard to ensure that they are effective, appropriate, consistent and user-friendly before they are integrated into the security concept [85, p. 32].

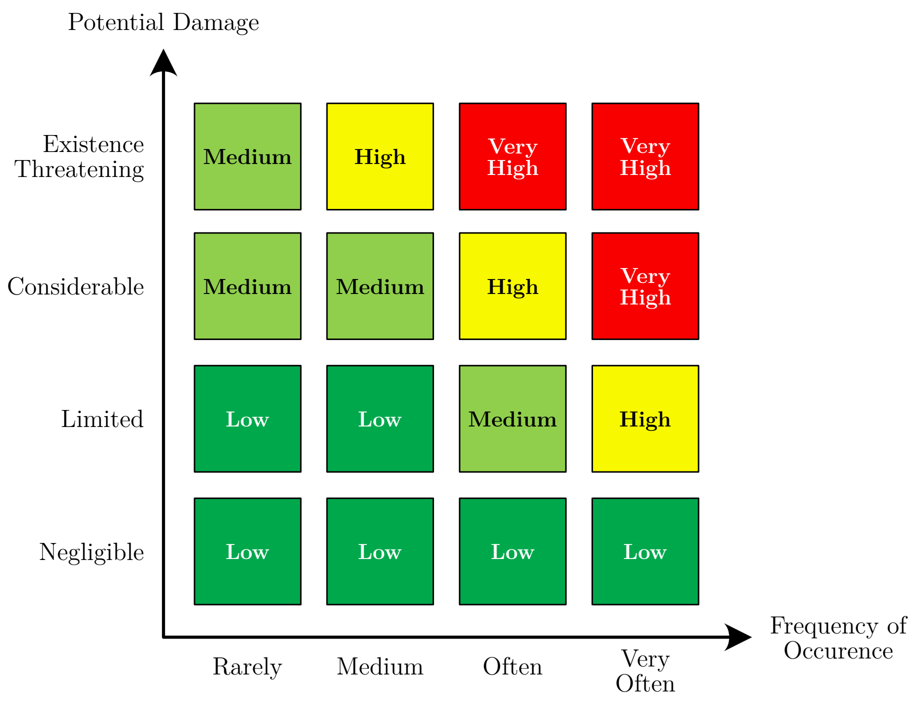


Figure 25: Risk classification matrix, based on [85, p. 22]

The BSI 200-3 specifies a flexible method that can be applied to both existing projects and those that are being planned. It guides the identification of security risks and their classification. Therefore, it is used in chapter “Risk Analysis with BSI 200-3” to find potential risks KNX installations could be exposed to, despite using KNXnet/IP Secure.

4. Analysis of KNXnet/IP Secure in ISO 22510:2019

The ISO 22510:2019 standard claims that KNXnet/IP Secure would provide confidentiality, mutual authentication, data integrity and resistance against replay attacks [6, pp. 101-102]. Only users with sufficient permissions are supposed to be able to access the provided services [6, p. 109]. In summary, KNXnet/IP Secure is meant to address the security shortcomings of KNXnet/IP. This chapter analyzes the standard to determine whether it fulfills its own security goals, which were previously explained in section “3.3 KNXnet/IP Secure in ISO 22510:2019”, as well as the current state of the art in cryptography. Furthermore, ISO 22510:2019 is checked for ambiguities and contradictions in the protocol specifications.

4.1. Unicast

In order to determine the security properties of the unicast protocol, it first has to be evaluated whether cryptographic primitives are applied correctly. Since KNXnet/IP Secure uses CCM for AEAD with a custom formatting and counter generation function, they are checked for their fulfillment of definitions 3.2 and 3.3 as required by the standard NIST SP 800-38C [3] and Jonsson’s security proof [35]. A violation of the requirements could indicate an issue that affects the confidentiality and authentication of the cipher mode. Furthermore, the design of the AKE is analyzed. This includes both the suitability of the selected cryptographic primitives and whether their composition constitutes a secure protocol for the establishment of a session key. The analysis takes a particular interest in the claimed security properties for the unicast protocol, which are confidentiality, mutual authentication, data integrity, resistance against replay attacks and dictionary attacks, as shown in section “3.3 KNXnet/IP Secure in ISO 22510:2019”. The evaluation also verifies whether the flaws found by Judmayer et al. [1] in early drafts of the protocol still exist in ISO 22510:2019. Additionally, it is checked if the unicast session and its access control fully address the security risks identified in KNXnet/IP, as previously discussed in subsection “3.1.6 Insecurities and Design Flaws in KNXnet/IP”. A formal analysis of the AKE is conducted with Cremer et al.’s eCK-PFS [68]. This model was chosen because it improves upon the work of Canetti and Krawczyk [66], and LaMacchia et al. [67] and permits proving if the protocol ensures PFS. Model checking with NuXMV is applied to formally verify whether the described behavior in ISO 22510:2019 matches the specified session FSM. The purpose is to identify discrepancies which could potentially affect the protocol’s security or indicate a quality problem in the writing of the standard. Based on the identified issues, recommendations are made for the improvement of the unicast protocol.

4.1.1. CCM Requirements

As previously explained in subsection “3.3.1 CCM” of the “Background” chapter, the ISO 22510:2019 standard does not explicitly define a formatting and counter generation function like NIST SP 800-38C does with its canonical ones. Both functions are implied by the B_0 and Ctr_i specified for each frame type. In the unicast protocol the frames `SESSION_RESPONSE`, `SESSION_AUTHENTICATE` and `SECURE_WRAPPER` make use of CCM. This results in the definition of the formatting function β as shown in figure 11 and the counter generation function π in figure 12. They have to satisfy the requirements of definition 3.3 in order to ensure that the security proof holds.

The first requirements is that the B_0 has to uniquely determine the nonce N. The B_0 and Ctr_i are equal in the 112 most significant bits for each respective frame type. This can be seen in definitions 3.38, 3.39 and 3.40. Comparing the Ctr_i for different frame types, it is evident that the 16 least significant bit always consist of `0xff`$\|i$. Since π is only supposed to take the nonce and block number as input, see figure 5, the nonce has to be $N = \mathrm{MSB}_{112}(Ctr_i)$, which is equivalent to $N = \mathrm{MSB}_{112}(B_0)$. Since this function uniquely determines the nonce N for a given B_0 the condition is satisfied.

The second requirement is that the formatted data uniquely determines P and A. Furthermore, β has to be prefix-free. The former part is fulfilled by the formatting of blocks $B_1, ..., B_n \leftarrow \mathrm{PAD}_{16}(a\|A\|P)$, as previously defined in equation 23. Since a is specified to have a fixed length of 16 bit and represents the length of A, and the length Q of P is either known or can be derived from $len(C) - T_{len}$, there exists a function that splits the given blocks $B_1, ..., B_n$ back into a, A and P. The a is given by the first 16 bit of B_1, the contained value determines the number of subsequent bytes that are A and the remaining bytes are the padded P. Removal of the padding is possible given knowledge about the length of P. Thus P and A are uniquely determined by the formatted data. For the latter part is has to be proven that β is prefix-free.

Proof 4.1. Let (N, P, A) and (N, P', A') be distinct input triples for β. Let the outputs be $\beta(N, P, A) = B_0, B_1, ..., B_r$ and $\beta(N, P', A') = B'_0, B'_1, ..., B'_r$. Assume that β is not prefix-free, then there exists a case where no $i \leq \min(r, r')$ can be found for which $B_i \neq B'_i$.

If (N, P, A) and (N, P', A') are distinct input triples because $P \neq P'$:

1. If P and P' have different lengths, then $Q \neq Q'$. Thus $B_0 \neq B'_0$, because $Q = \mathrm{LSB}_{16}(B_0)$ and $Q' = \mathrm{LSB}_{16}(B'_0)$. This covers the case when P is a prefix of P' or vice versa.
2. If P and P' are of equal length, with $P \neq P'$, then there has to exist a B_i and B'_i with $i \leq \min(r, r')$ for which $B_i \neq B'_i$ because $B_1, ..., B_r \leftarrow \mathrm{PAD}_{16}(a\|A\|P)$ and $B'_1, ..., B'_r \leftarrow \mathrm{PAD}_{16}(a'\|A'\|P')$.

If (N, P, A) and (N, P', A') are distinct input triples because $A \neq A'$:

1. If A and A' have different lengths, then $a \neq a'$. Thus $B_1 \neq B'_1$, because $B_1, ..., B_r \leftarrow \mathrm{PAD}_{16}(a\|A\|P)$ and $B'_1, ..., B'_r \leftarrow \mathrm{PAD}_{16}(a'\|A'\|P')$. This covers the case when A is a prefix of A' or vice versa.

2. If A and A' are of equal length, with $A \neq A'$, then there has to exist a B_i and B'_i with $i \leq \min(r, r')$ for which $B_i \neq B'_i$ because $B_1, ..., B_r \leftarrow \text{PAD}_{16}(a\|A\|P)$ and $B'_1, ..., B'_r \leftarrow \text{PAD}_{16}(a'\|A'\|P')$.

In all cases where (N, P, A) and (N, P', A') can be distinct input triples there exists an $i \leq \min(r, r')$ for which $B_i \neq B'_i$. Hence, by proof of contradiction, β is prefix-free.

For the different notation of the second requirement by Jonsson [35], it can also be shown that when the input triples are expected to contain nonces N and N' respectively, with $N \neq N'$, then $B_0 \neq B'_0$ because $N = \text{MSB}_{112}(B_0)$ and $N' = \text{MSB}_{112}(B'_0)$. Thus, in all cases were the input triples can be distinct, β is prefix-free.

The third requirement is that B_0 may not be equal to any of the counter blocks Ctr_i across all invocation of CCM under the same key. This is evident from the constructions of B_0 and Ctr_i as described in subsection "3.3.1 CCM", and definitions 3.38, 3.39 and 3.40. `SESSION_RESPONSE` and `SESSION_AUTHENTICATE` frames have a fixed length and their B_0 as well as Ctr_0 are constants. The $B_0 \neq Ctr_0$ because $\text{LSB}_{16}(B_0) \neq \text{LSB}_{16}(Ctr_0)$, as shown in definition 3.38 and 3.39. For `SECURE_WRAPPER` frames the length of the plaintext Q and the counter i are contained within the 16 least significant bits of B_0 and the Ctr_i respectively. ISO 22510:2019 requires that the length of the plaintext does not exceed 65279 bytes, which is `0xfeff` in hexadecimal [6, p. 107]. Additionally, the counter is limited to a maximum of $2^8 - 1 = 255$ or `0xff`, which practically limits the plaintext length to 255×16 byte $= 4080$ byte [6, p. 107]. Based on definition 3.40 the $Q = \text{LSB}_{16}(B_0)$ and $\texttt{0xff}\|i = \text{LSB}_{16}(Ctr_i)$. Since Q cannot exceed a value of 4080, which is `0x0ff0` in hexadecimal, it follows that $B_0 \neq Ctr_i$.

KNXnet/IP Secure fulfills all three requirements from definition 3.3 as originally defined in NIST SP 800-38C. However, it should be considered that the different formatting and counter generation functions likely require manufacturers to implement their own versions of CCM. This can pose an increased risk for subtle issues in the implementation, if the code has not been extensively reviewed by experts in cryptography and application security. The concerns of Rogaway in [38] about the non-binding specification of the canonical formatting and counter generation function in NIST SP 800-38C appear to be justified.

It is also necessary to evaluate if definition 3.2 is fulfilled by KNXnet/IP Secure's usage of CCM. It requires that the nonce N is never repeated under the same key for different input data. This requirement exists, because nonce reuse can break the confidentiality of the encrypted plaintext. When N is repeated in invocations of CCM under the same key, this leads to π generating the same $Ctr_0, Ctr_1, ..., Ctr_m$, which results in an identical key stream S and S_0. This can be seen in figure 5. Since encryption is accomplished through $C \leftarrow (P \oplus \text{MSB}_{\text{len}(P)}(S))\|(T \oplus \text{MSB}_{T_{len}}(S_0))$, CCM has the same issue as the CTR cipher mode. The impact on the confidentiality can be explained based on a simplified example. Consider two blocks of plaintexts P_i and P'_i, which are supposed to be encrypted by the same S_i through $C_i = P_i \oplus S_i$ and $C'_i = P'_i \oplus S_i$. Both equation can be transformed into $S_i = P_i \oplus C_i$ and $S_i = P'_i \oplus C'_i$ by applying the XOR of P_i and P'_i respectively. Therefore, $P_i \oplus C_i = P'_i \oplus C'_i$

which is equivalent to $P_i \oplus C_i \oplus C'_i = P'_i$. An adversary that is able to monitor the communication will have access to C_i and C'_i. If they also have knowledge of one plaintext, P_i in the simplified example, they can decrypt all other ciphertexts under the same nonce and key, up to the length of the known plaintext. Even if such a plaintext is not known, information about it are revealed, because $C_i \oplus C'_i = P_i \oplus P'_i$. The XOR of the ciphertexts will contain zeros whenever bits in the unknown plaintexts are equal. The frequency of such matches may allow inferring information about the content.

In `SESSION_RESPONSE` frames the B_0 is constant, as shown in definition 3.38. Furthermore, the key that is being used by the server for CCM is the static device authentication code. Therefore, the nonce is constant under the given key, as $N = \text{MSB}_{112}(B_0)$, which violates the requirement. However, `SESSION_RESPONSE` applies CCM only for authentication of the KNXnet/IP Secure header, the secure session identifier and $X \oplus Y$. All of its content is unencrypted, besides tag T. Since T is never transmitted in the clear, an adversary should not be able to obtain this intermediate value of CCM. Thus, using a known tag to decrypt $T \oplus \text{MSB}_{T_{len}}(S_0)$ does not appear to be possible, despite the reuse of the nonce. Furthermore, T contains the CBC-MAC, which has been computed with AES. Since it is a PRF, frequency analysis should not be able to recover meaningful information from different ciphertexts. Even under the assumption that an adversary would be able to retrieve the T and S_0, an adversary would not be able to perform a MAC forgery with a length extension attack, because B_1 contains the length of the authenticated data and the content of the frame has a fixed size. Therefore, the nonce reuse does not appear to have a negative effect in this specific instance.

In `SESSION_AUTHENTICATE` frames the B_0 is constant, as shown in definition 3.39. The key that is being used by the client for CCM is the password hash for the user it wants to authenticate itself as. Therefore, the nonce is constant under the given key, as $N = \text{MSB}_{112}(B_0)$, which violates the requirement. For the same reasons as with the `SESSION_RESPONSE` this does not appear to be an issue. Additionally, the `SESSION_AUTHENTICATE` is only allowed to be transmitted when encapsulated in a `SECURE_WRAPPER` [6, p. 129].

In `SECURE_WRAPPER` frames the B_0 consists of the sequence number, KNX serial number, message tag and plaintext length, as shown in definition 3.40. As previously explained in "3.3.2.4 Authenticated Key Exchange", the message tag is required to be `0x0000` for unicast communication and the KNX serial number is assumed to be a fixed value for a given device. Hence, only the sequence number is a variable value in the nonce, as $N = \text{MSB}_{112}(B_0)$. Each party in the session has a sequence number that is initialized to zero and incremented with each frame that they are sending [6, p. 122]. This is the sequence number included in B_0, which is 48 bit long, as shown in definition 3.40. The recipient has to store the last valid sequence number that they have received in the session. Incoming frames are only supposed to be accepted "[…] if the sequence number is greater than the sequence number of the previously successfully received frame on the same connection" [6, p. 122]. It could happen that a KNXnet/IP Secure session is active long enough for the sequence number to reach its maximum. ISO 22510:2019 does not specify how the sender should handle this situation, it only states that under the assumption that one million frames are being sent per second the overflow would occur after 9 years [6, p. 122]. Depending on the maximum data rate supported by the medium and network interface, the queue size of the devices, and their processing speed, the required time could also be an order of magnitude less. Since the specification does not state what the resulting value of the sequence number is after an overflow, it is assumed that it wraps around to zero. This would cause a nonce reuse under the same session key. The same issue would also occur if the sequence number would be stuck at the maximum value. While the recipient would not accept such frames, because the sequence number is smaller (or equal) to the stored one, it does not prevent the sender from creating them. As a consequence, the confidentiality would be affected. An adversary could record frames sent at the beginning of the session and after the overflow, so that they have access to pairs of frames with the same nonce. It is not necessary for them to store all transmitted frames to find a nonce reuse. The session would timeout shortly after the overflow, because the recipient does not receive any valid frames anymore. Servers would close the session after one minute, see figure 19 with action A4 in the "Authenticated" state and event E06. This is why storing the more than 2^{48} frames is not required, since in this hypothetical scenario the frames with reused nonces would only be a small fraction of the sequence number space. Even if a real occurrence of a sequence number overflow is not considered likely by the KNX Association, the ISO 22510:2019 should specify how this edge case is supposed to be handled, because it affects the session's confidentiality.

4.1.2. Authenticated Key Exchange and Session

The unicast protocol uses CCM, X25519, and PBKDF2 with HMAC-SHA-256 as its cryptographic primitives for the AKE and session. CCM is one of the approved cipher modes listed in BSI TR-02102-1 [44] and NIST SP 800-131A [45], standardized by NIST SP 800-38C [3]. Besides the requirements checked in the previous section, the BSI additionally recommends a tag length of at least 64 bit [44, p. 24]. KNXnet/IP Secure fulfills this with a fixed tag length of 128 bit, as can be seen in definitions 3.38, 3.39 and 3.40, since the tag length is equal to the length of the MAC field. CCM is a suitable cipher mode for AEAD, when applied correctly. As shown in the previous section, nonce reuse could occur in the unicast protocol of KNXnet/IP Secure.

While Curve25519 is neither included in BSI TR-03111 [46] nor in NIST SP 800-56A [126], it is an elliptic curve that is believed to be secure, as discussed in subsection "3.2.2 Elliptic Curve Cryptography and Curve25519". Wireguard [127] and Signal [128] use Curve25519 in their protocols as well. There have been discussions among cryptographers regarding the optional input validation for X25519 [113], [114]. The criticism of the design is not justified for the usage of X25519 in AKE protocols. If an AKE protocol ensures mutual authentication and data integrity, then an adversary would not be able to force a shared secret of zero, as they cannot manipulate a public key to be a low order point. When a legitimate client generates their public key as specified in RFC 7748 [7], then it would not produce such a point either. Hence, this would only happen if a client, that can authenticate itself, deliberately sends a low order point as public key. In this case the client would only compromise confidentiality in its own session. If the AKE cannot ensure mutual authentication and data integrity, then the protocol is generally broken and susceptible to MitM attacks. Rejecting certain inputs would not prevent this problem. Therefore, X25519 is a suitable choice for asymmetric cryptography for ECDH in an AKE.

KNXnet/IP Secure relies on PBKDF2-HMAC-SHA-256 for password-based key derivation and subsequent storage of the password hash / key. While the specific application of PBKDF2 for the authentication in the AKE is evaluated later in this section, it is no state-of-the-art algorithm for password hashing. Unlike scrypt [129] or Argon2 [130], [131], PBKDF2 is not designed to withstand highly parallelized attacks with ASICs or GPUs. Although, with a random salt per user, a high enough iteration count and a strong password, it could still be computationally infeasible to brute force. However, PBKDF2 does not appear to be a suitable building block for authentication in an AKE though, especially when digital signatures algorithms based on elliptic curves would be an option. If the authentication in an AKE is based on symmetric cryptography, more than one party has to know the respective key. This has security implication, in case one of the parties, that is meant to verify that another party's frames are authentic, gets compromised.

The AKE of the unicast protocol has been previously described in "3.3.2.4 Authenticated Key Exchange". A simplified description of the most important steps is shown in figure 26. The initial frame is a `SESSION_REQUEST` sent by the client that contains its public key X. While this frame is not encrypted or authenticated, this is not a design error. The client authenticates itself to the server at a later point with a wrapped `SESSION_AUTHENTICATE`, which is meant to ensure that the client is not being impersonated. However, an issue in the ISO 22510:2019 specification is that the `SESSION_REQUEST` format is specified as variable length, because supposedly the HPAI could be of variable length [6, p. 125]. This is contradictory to the description of the validation steps that only permit `SESSION_REQUEST`'s received over TCP with a route back HPAI for TCP [6, p. 126]. Thus, the frame has to have a fixed length of 46 byte [6, p. 126]. Additionally, the order of the validation steps is incorrect. The length check has to occur before any evaluation of the HPAI content, otherwise malformed frames could get misinterpreted.

Judmayer et al. [1] claimed that in AN 159/13 v02 there was a risk of a denial of service (DoS) attack. Since in this version UDP was permitted and the first frame in the handshake could be spoofed, they argued that this would allow to both initiate computationally costly operations on the server for the ECDH and the adversary could exhaust the number of sessions that can be simultaneously active [1, pp. 6-7]. Since the current standard requires a TCP connection and the client has to respond within 10 seconds to the `SESSION_REQUEST` or the session is terminated [6, p. 114], this scenario would not apply in the same way to devices conforming with ISO 22510:2019. Although, general attack vectors for DoS, such as saturating the link with requests or attempting to interrupt existing connections by spoofing TCP resets, still exist.

Furthermore, bugs in the implementation may allow effective DoS attacks. Chapter "Analysis of Certified Devices" provides a real world example caused by the incorrect length validation of `SESSION_REQUEST` frames.

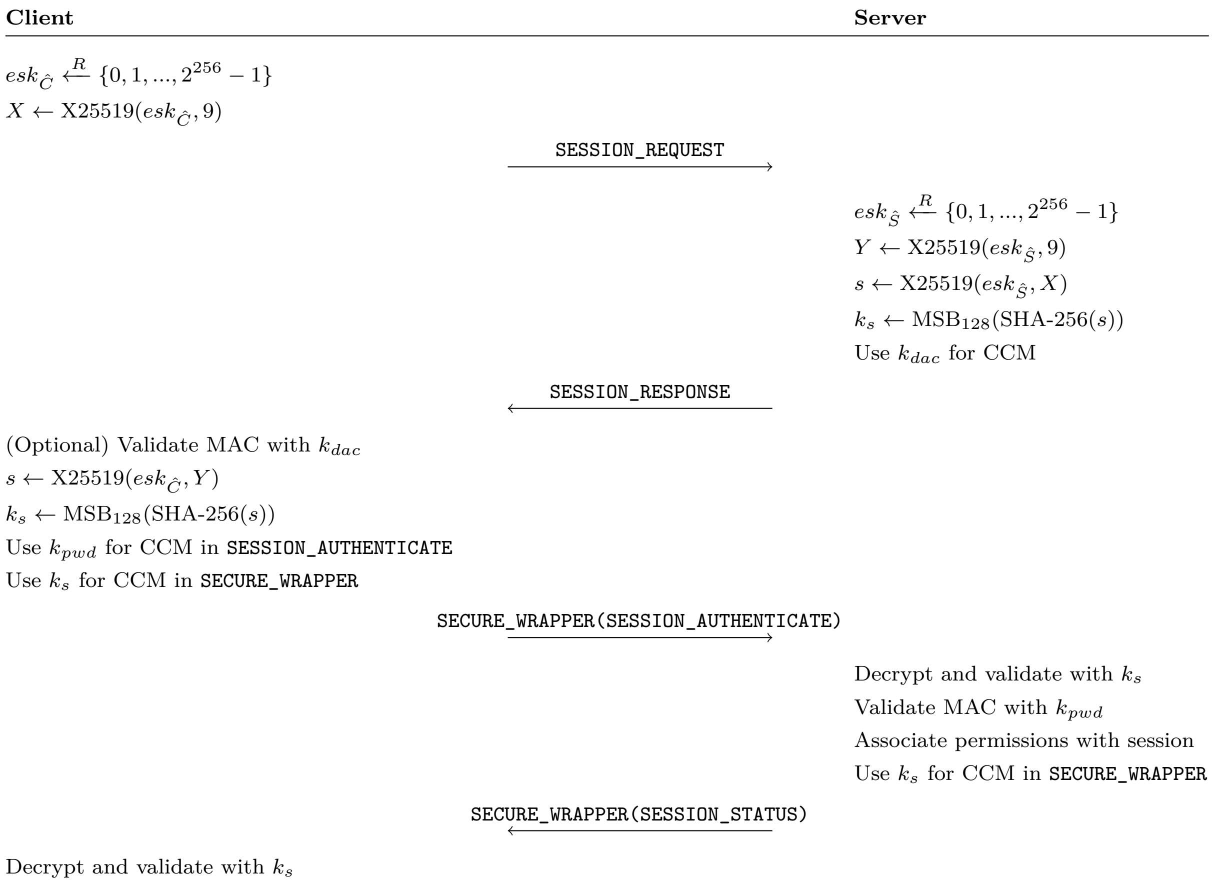

Client
Server
$esk_{\hat{C}} \xleftarrow{R} \{0, 1, ..., 2^{256} - 1\}$
$X \leftarrow \mathrm{X25519}(esk_{\hat{C}}, 9)$
SESSION_REQUEST
$esk_{\hat{S}} \xleftarrow{R} \{0, 1, ..., 2^{256} - 1\}$
$Y \leftarrow \mathrm{X25519}(esk_{\hat{S}}, 9)$
$s \leftarrow \mathrm{X25519}(esk_{\hat{S}}, X)$
$k_s \leftarrow \mathrm{MSB}_{128}(\text{SHA-256}(s))$
Use k_{dac} for CCM
SESSION_RESPONSE
(Optional) Validate MAC with k_{dac}
$s \leftarrow \mathrm{X25519}(esk_{\hat{C}}, Y)$
$k_s \leftarrow \mathrm{MSB}_{128}(\text{SHA-256}(s))$
Use k_{pwd} for CCM in SESSION_AUTHENTICATE
Use k_s for CCM in SECURE_WRAPPER
SECURE_WRAPPER(SESSION_AUTHENTICATE)
Decrypt and validate with k_s
Validate MAC with k_{pwd}
Associate permissions with session
Use k_s for CCM in SECURE_WRAPPER
SECURE_WRAPPER(SESSION_STATUS)
Decrypt and validate with k_s

Figure 26: Simplified depiction of the AKE
Happy path without the frame construction or full validation

When the server receives a valid `SESSION_REQUEST` it can compute its own private key $esk_{\hat{S}}$ and public key Y for the session. The server determines the shared secret with the received public key X of the client, as described in "3.3.2.4 Authenticated Key Exchange", and depicted in figures 26 and 8. The session key k_s is derived with $k_s \leftarrow \mathrm{MSB}_{128}(\text{SHA-256}(s))$. This key derivation appears to be suitable, since SHA-256 is a cryptographically secure hash function, the server uses an ephemeral key and only a single session key has to be derived. Hence, a more elaborate key derivation function, such as HKDF [132], is not strictly required. This design change was introduced after AN 159/13 v04, which used to truncate the shared secret directly to determine the session key [19, p. 7]. Since the shared secret is not a uniformly random byte sequence, this not advisable. This issue was first criticized by Judmayer et al. in the design of AN 159/13 v02 [1, p. 6].

The server answers the `SESSION_REQUEST` with a `SESSION_RESPONSE` only if it has the available resources for an additional session [6, p. 128]. This prevents an DoS attack on the client that was possible in v02 and v04, which was discovered by Judmayer et al. [1, p. 6]. These previous draft versions required the server to respond in an unauthenticated frame to indicate an error [19, p. 18], which an adversary could forge to prevent a legitimate client from connecting to a server. In v06 and ISO 22510:2019 errors are only indicated with a `SESSION_STATUS` frame, wrapped in a `SECURE_WRAPPER`, once the session is established.

When the server is able to create another session and has reserved a session identifier, it can send a `SESSION_RESPONSE`. In ISO 22510:2019 the `SESSION_RESPONSE` authenticates the header, session identifier and $X \oplus Y$, as shown in definition 3.38. The authentication of $X \oplus Y$ instead of their concatenation is an unusual design choice. A theoretical risk of a MAC forgery was identified by Judmayer et al. [1, pp. 7-8]. According to them the MAC that authenticates $X \oplus Y$ could be reused to impersonate the server, if for a `SESSION_REQUEST` containing X' the adversary would be have knowledge of a Y' and its matching private key, for which $X \oplus Y = X' \oplus Y'$. Since the ECDLP and related problems apply, the adversary cannot determine the private key for such a Y', they would already have to know it. Exhaustively generating key pairs is not an option, since the number of points on Curve25519 are $8 \times 2^{252} + 27742317777372353535851937790883648493$ [42, p. 214]. The chance of an adversary picking a random private key, for which the resulting public key Y' fulfills $X \oplus Y = X' \oplus Y'$, is approximately zero. It is even more unlikely than finding a collision in the resulting 128 bit MAC. Therefore, this is not a practicable attack and this does not even consider the varying session identifiers. Despite the low probability of an actual impact on the authentication, the concatenation of X and Y would be a preferable design choice.

Application Notes v04 and v06 suggested that Diffie et al.'s STS protocol [48] would be comparable to KNXnet/IP Secure's AKE [19, p. 7], [49, p. 34]. It is unclear why this reference was included, because the AKEs are quite different besides their application of (EC)DH. KNXnet/IP Secure uses pre-shared symmetric keys for the authentication, while the STS protocol relies on certificates and signatures. While the reference to the STS protocol were removed in ISO 22510:2019, it shows that the choice of symmetric over asymmetric cryptography for the authentication was deliberate, despite the designers being aware of other, safer approaches. The content of the `SESSION_RESPONSE` is authenticated with the device authentication code, or FDSK, when the server is in factory default settings. These keys have to be known by all potential clients so that they can verify that the frames are authentic. Judmayer et al. pointed out the obvious issue of device compromise with symmetric keys for authentication [1, p. 4]. If any of clients that store the server's device authentication code are compromised and the adversary is able to extract this key, they can impersonate the server. The problems are even more severe in ISO 22510:2019 as the device authentication code is "[...] derived from a user chosen shared secret" with PBKDF2(p, "device-authentication-code.1.secure.ip.knx.org", $65536, 128$) [6, p. 135]. A static salt is used, which means that if the same password is used for different servers, they derive the same device authentication code. As explained in "3.2.3 Key Derivation with PBKDF2-HMAC-SHA-256", this is against best practice. The compromise of one server could potentially break the authentication of other servers. Furthermore, the standard claims that the symmetrical keys would be protected against offline dictionary attacks [6, p. 144]. This is categorically false, since offline attacks can be conducted and it is possible to precompute a dictionary with commonly used passwords, since the salt is static. All that is necessary for an offline attack is to record the first two frames in an AKE with the target server. Since the content in the `SESSION_RESPONSE` frame is only authenticated and not encrypted, and the public key of the client is known from the `SESSION_REQUEST`, all inputs for the CCM encryption for the `SESSION_RESPONSE` are known, except for the key. The adversary can guess passwords, derive the keys with PBKDF2 and check whether one of the keys allows to successfully verify the MAC field in the `SESSION_RESPONSE`. The security of the authentication hinges on the complexity of the password and the computational resources of the adversary, as they determine whether it is practically possible to find the key. An evaluation is conducted in chapter "Device Management with the ETS5" with a modified version of the GPU-accelerated password cracking software hashcat [133].

ISO 22510:2019 claims that KNXnet/IP Secure unicast would provide mutual authentication [6, pp. 101-102, p. 144]. However, it is optional for clients to validate the MAC in the server's `SESSION_RESPONSE` [6, pp. 127-128]. If the authentication of the server is not mandatory, then the protocol does not provide mutual authentication. An adversary can impersonate any server to a client that does not verify that the `SESSION_RESPONSE` frames are authentic.

With the `SESSION_RESPONSE` frame received, the client is able to determine the shared secret and session key, based on the public key of the other party. The client has to authenticate itself to the server with a password hash and associated user ID, where the user ID determines the level of permissions granted by the server [6, pp. 129-130]. This is done through a `SESSION_AUTHENTICATE` frame, using a password hash as key to authenticate its content with CCM. The frame is wrapped in a `SECURE_WRAPPER`, encrypted and authenticated with the session key. Each password hash is derived from a password p with PBKDF2(p, "user-password.1.secure.ip.knx.org", 65536, 128) [6, p. 135]. ISO 22510:2019 states that "For practical reasons, a user may choose to opt for the same password set in all devices of his installation project" [6, p. 130]. While this is technically true, it reads like a suggestion. This would be ill-advised, especially give that the key derivation uses a static salt that is identical for all users and devices. Therefore, the password reuse across devices would result in the same password hash being generated. This issue already existed in AN 159/13 v02, as pointed out by Judmayer et al. [1, p. 4], despite using a different approach for the key derivation. As a consequence, the compromise of password hashes from one server would potentially allow an adversary to impersonate the users to another server, where the password has been reused. The static salt is a design flaw known as CWE-760 [134].

After a factory reset or before the initial configuration, all password hashes are derived from the empty password and are thus a well-known static key, as explained in

“3.3.2 Unicast”. In this situation the server cannot perform an actual authentication of the client, as the standard admits [6, p. 122]. Hence, it is advisable to configure the device before installing them in the network. Otherwise, an adversary could authenticate itself with the known key and change settings or use the provided services. While it is possible to perform a factory reset with physical access or overwrite the configuration, it is against best practice that the default credentials are hard-coded, see CWE-798 [135]. It is unclear, why ISO 22510:2019 does not require a random, device specific password set by the manufacturer, similar to the FDSK.

An adversary could attempt to gain knowledge about a password hash used by a client, by impersonating a server that the client intends to communication with. Due to the flawed authentication, the adversary could either send a `SESSION_RESPONSE` with arbitrary content in the MAC field, if the client does not validate it, or attempt to determine the device authentication code of the legitimate server through password cracking, as previously described. If either approach is successful, the adversary is able to complete an AKE with the client. Since the adversary obtains the actual session key k_s, because they pass as a legitimate party, they are able to decrypt the `SECURE_WRAPPER` that contains the `SESSION_AUTHENTICATE` frame. Therefore, the adversary has access to the MAC field in the `SESSION_AUTHENTICATE` frame, that authenticates the content of the frame with CCM, using a password hash as key. Similarly to the password cracking for the device authentication code, the adversary has knowledge of all inputs that were used to compute the MAC field, except for the key. An offline attack can be conducted in the same fashion, by calculating the password hashes for potential passwords and checking if the MAC can be successfully verified with one of them. Again, the practical feasibility of the attack depends on the password complexity and available computational resources. It is evident that the authentication in KNXnet/IP Secure, as specified by ISO 22510:2019, cannot be considered secure from a theoretical standpoint. Its practical security highly depends on the chosen passwords and the clients voluntary authentication of the server. How the passwords are generated with the ETS is explained in “Device Management with the ETS5”.

When a server is able to validate the MAC, contained in the wrapped `SESSION_AUTHENTICATE` frame, with the password hash stored in `PID_PASSWORD_HASHES` for the given user ID, it determines whether management level access or user level access is provided. This was previously explained in “3.3.2.4 Authenticated Key Exchange” and “3.3.2.6 Access Control”. KNXnet/IP Secure implements access control based on these two permission levels and the settings of `PID_SECURED_SERVICE_FAMILIES`. ISO 22510:2019 restricts the access to device management services, when `PID_SECURED_SERVICE_FAMILIES` requires KNXnet/IP Secure to be used. Only management level access is allowed, this prevents configuration through plain KNXnet/IP and user level access. Additionally, the configuration of KNXnet/IP Secure parameter objects requires KNXnet/Data Secure and knowledge of the tool key [6, p. 129, p. 133]. Hence, all KNXnet/IP Secure device need to support Data Secure to protect the cEMI frames for the configuration of the server. Furthermore, if `PID_SECURED_SERVICE_FAMILIES` requires KNXnet/IP Secure for tunneling, the service may not be accessed through plain KNXnet/IP. Thus, access control addresses the concerns raised in “3.1.6 Insecurities and Design Flaws in KNXnet/IP", although the design flaws in the authentication undermine its goal of restricting access to authorized users. Device discovery remains unrestricted with KNXnet/IP Secure, thus an adversary can still learn detailed information about the devices that permit to identify the specific model and manufacturer without any authentication. This may help an adversary to identify devices that are known to be vulnerable.

Once the server informs the client about the successful authentication with a `SECURE_WRAPPER` containing the `SESSION_STATUS` frame, the AKE is completed. Further communication in the session is only conducted through frames encapsulated in `SECURE_WRAPPER` that are encrypted and authenticated with the session key. Besides the theoretical nonce reuse, no further flaws have been found in the session communication.

The evaluation of the security properties defined in "3.2.4 Security Properties for Cryptographic Protocols" had the following results:

- **Confidentiality** is provided during the AKE under the assumption that both parties are honest and therefore the adversary does not actively interfere. The ECDH ensures through the ECDLP that an adversary is unable to determine the private keys of both parties and thus the session key is secret. CCM is used for AEAD. However, during the session it is theoretically possible that a nonce reuse occurs, due to a sequence number overflow. Thus, ISO 22510:2019's goal of confidentiality is not generally fulfilled for the session. Furthermore, if the adversary is permitted to actively attack by impersonating a party, then the confidentiality is broken, as they are able to learn the session key.
- **Data integrity** is supposed to be provided by the application of CCM with either a device authentication code, password hash or session key. As previously explained, the client is not required to validate `SESSION_RESPONSE` frames. Additionally, an adversary is potentially able to determine device authentication codes and password hashes, hence able to forge frames that pass validation. Thus, it is not ensured that altered or injected messages can be detected under all circumstances. This goal of ISO 22510:2019 is not fulfilled.
- **Message authentication** is not ensured, as it is based on data integrity. Since an adversary could learn device authentication codes or password hashes, they would be able to forge frames that appear to originate from a particular party. Thus, the identity of another party cannot be accurately verified, as the legitimate and forged frames cannot be distinguished. Therefore, an adversary can impersonate parties. Additionally, the password-based key derivation uses static salts, thus password reuse leads to the same keys being generated. As a consequence, it is not always possible to distinguish the origin of a message. For instance, servers could share the same device authentication code, resulting in frames that could not be attributed to a single party.
- **Entity authentication** is not ensured, as the adversary is potentially able to impersonate parties.

- **Mutual authentication** is a goal of ISO 22510:2019 that is also not fulfilled, since entity authentication is not ensured. Furthermore, this is already conceptually broken by making the MAC validation of `SESSION_RESPONSE` frames optional.
- **Non-repudiation** is not ensured, since multiple parties are required to have knowledge of the device authentication codes and password hashes to authenticate each other. As a consequence, any of those parties could in theory have created a particular frame. For instance, it is not possible to determine if a `SESSION_RESPONSE` frame was created by a server or by one the clients that know its device authentication code, just by considering the information contained in the frame. A similar issue occurs when passwords have been reused and the origin of the message is unclear, because more than one party shares the same key for the authentication.
- **Resistance against replay attacks** is provided by the sequence numbers integrated into the `SECURE_WRAPPER`. A recipient does not accept replayed frames, even in the edge case when the sender has a sequence number overflow. Frames in the AKE cannot be effectively replayed, since ECDLP prevents an adversary from learning the private from the public key. Hence, it is possible to replay a `SESSION_RESPONSE` frame, but the adversary would not be able to derive the session key. This goal of ISO 22510:2019 is fulfilled.
- **Perfect forward secrecy** is ensured, since both parties are required to generate new key pairs for ECDH in every session [6, p. 121]. These ephemeral keys are unrelated to the device authentication codes and password hashes, thus a compromise of the long-term keys does not affect the confidentiality of past session keys that have been established between honest parties.
- **Resistance to known session key attack** is also fulfilled, for the same reason as PFS. Since every session requires fresh ephemeral key pairs by both parties, the session key will be different to past sessions. Thus, knowing a particular session key does not provide an advantage for attacking other sessions. The likelihood of the exact same choice of key pairs repeating for both parties is negligible.
- **Resistance against key compromise impersonation** is not ensured. If the adversary has knowledge of the long-term keys of a party, this includes their device authentication code(s) and password hashes. Since these keys are used to authenticate their communication partners, the adversary can impersonate them.
- **Resistance against unknown key-share attack** is not ensured, due to the potential password reuse. An unknown key-share could occur, if there are two servers $\hat{S}_1$ and $\hat{S}_2$ that have the same device authentication codes and password hashes. When a client intends to communication with server $\hat{S}_1$, the adversary could in theory redirect the communication to server $\hat{S}_2$, while the client believes it communicates with $\hat{S}_1$. Since the device authentication code and password

hash match, both client and server would not notice the incorrect communication partner.

- **Implicit key authentication** is provided, because ECDH ensures through the ECDLP that only the client and server performing the AKE have knowledge of the session key among honest parties.
- **Key confirmation** is ensured, since both parties are expected to derive the session key and prove knowledge of it. The client has to apply CCM with the session key to encrypt the `SESSION_AUTHENTICATE` frame in the `SECURE_WRAPPER`. The server is able to verify that the session key was used by decrypting and validating the frame with said key. Confirmation of the successful establishment of the session is sent by the server with an encrypted `SESSION_STATUS` in a `SECURE_-WRAPPER`. The client can also verify that the server has knowledge of the session key by decrypting and validating the encapsulated frame.
- **Explicit key authentication** is fulfilled since both implicit key authentication and key confirmation hold.
- **Key freshness** is fulfilled, as ISO 22510:2019 requires both parties to generate new key pairs for every session [6, p. 121].
- **Key control** is not fulfilled, because ISO 22510:2019 does not require the optional input validation for X25519. Therefore, a party could deliberately send a low order point to force the shared secret to be zero. However, this is not practical attack, as previously explained.
- **Identity hiding** is not fulfilled, since a passive adversary can observe the `SESSION_RESPONSE` of a server and attempt to infer its device authentication code.

In summary, the unicast communication of KNXnet/IP Secure fails to fulfill a majority of the goals stated in ISO 22510:2019 from a theoretical standpoint. It does not provide confidentiality, data integrity, mutual authentication, and resistance against online and offline attacks under all circumstances. Only resistance against replay attacks could be confirmed among the intended goals. Furthermore, several security properties that are generally expected of a modern AKE protocol were not achieved. Since both theoretical and practical attacks are possible, due to severe design flaws in the authentication, it cannot be considered a secure protocol based on the current state of the art in cryptography. However, the feasibility of offline attacks against the authentication depends on the password complexity.

4.1.3. Formal Analysis with eCK-PFS

The unicast protocol of KNXnet/IP Secure is formally analyzed in the eCK-PFS model as defined in "3.2.5 eCK-PFS Model".

Definition 4.1 (Client). Let a client $\hat{C}_i \in \mathcal{P}$ be a party that posses the following key material.

- **Long-term keys**: Let $k_{pwd} \overset{R}{\leftarrow} \{0,1\}^\lambda$ be a password hash, $uid \in [1,127]$ a user ID and $\hat{S}_j \in \mathcal{P}$ a server. Let $pwd = (k_{pwd}, uid, \hat{S}_j)$ be a triple that associates a $\hat{S}_j$ with a k_{pwd} and uid, which can be used to authenticate the user to the $\hat{S}_j$. Let $PWD_{\hat{C}_i} = \{pwd_1, ..., pwd_n\}$ be the first set of long-term keys for $\hat{C}_i$, which are revealed by a corrupt($\hat{C}_i$) query.

  Let $k_{dac} \overset{R}{\leftarrow} \{0,1\}^\lambda$ be a device authentication code for a server $\hat{S}_j \in \mathcal{P}$. Let $dac = (k_{dac}, \hat{S}_j)$ be a tuple that associates the k_{dac} with a $\hat{S}_j$, which can be used to authenticate it. Let $DAC_{\hat{C}_i} = \{dac_1, ..., dac_m\}$ be the second set of long-term keys for $\hat{C}_i$, which are revealed by a corrupt($\hat{C}_i$) query. Since clients are not required to validate the server's device authentication code, the $DAC_{\hat{C}_i}$ could also be empty.

- **Ephemeral keys**: Let $esk_{\hat{C}_i} \overset{R}{\leftarrow} \{0,1,...,2^{256}-1\}$ be the ephemeral secret key and $epk_{\hat{C}_i} \leftarrow \text{X25519}(esk_{\hat{C}_i}, 9)$ the associated public key of $\hat{C}_i$ in a session s. These keys are revealed by the ephemeral-key(s) query. The $epk_{\hat{C}_i}$ of a $\hat{C}_i$ was previously referred to as X.

Definition 4.2 (Server). Let a server $\hat{S}_i \in \mathcal{P}$ be a party that posses the following key material.

- **Long-term keys**: Let $k_{pwd} \overset{R}{\leftarrow} \{0,1\}^\lambda$ be a password hash and $uid \in [1,127]$ a user ID. Let $pwd' = (k_{pwd}, uid)$ be a tuple that associates a k_{pwd} with a uid, which can be used to authenticate a client $\hat{C}_i$ through its user. Let $PWD_{\hat{S}_i} = \{pwd'_1, pwd'_2, ..., pwd'_n\}$ be the first set of long-term keys for $\hat{S}_i$, which are revealed by a corrupt($\hat{S}_i$) query.

  Let $k_{dac} \overset{R}{\leftarrow} \{0,1\}^\lambda$ be the device authenticate code for $\hat{S}_i$. This is another long-term key that is revealed by a corrupt($\hat{S}_i$) query.

- **Ephemeral keys**: Let $esk_{\hat{S}_i} \overset{R}{\leftarrow} \{0,1,...,2^{256}-1\}$ be the ephemeral secret key and $epk_{\hat{S}_i} \leftarrow \text{X25519}(esk_{\hat{S}_i}, 9)$ the associated public key of $\hat{S}_i$ in a session s. These keys are revealed by the ephemeral-key(s) query. The $epk_{\hat{S}_i}$ of a $\hat{S}_i$ was previously referred to as Y.

The information about associated user IDs and servers are considered part of the long-term keys, as the specified protocol requires this information alongside the password hashes and device authentication codes for the authentication. Additionally, this does not change the advantage $\text{Adv}_{\mathcal{A}}^{\text{eCK-PFS}}(\lambda)$ for the adversary $\mathcal{A}$. It is possible to conduct an attack game without the associated information, but this would always require to describe the iteration through clients, servers and/or user IDs, for which the revealed key can be used. Therefore, this is mainly for a compact description of the attack games and has no effect on the outcome.

This analysis in eCK-PFS assumes the best case scenario for ISO 22510:2019, where clients do not skip the authentication of servers and passwords are not reused. If it can be shown that the unicast protocol is insecure in the model under these condition, then it will also be insecure when the server authentication is skipped or passwords are reused. Attack games that prove this can be found in section "A.1 Supplementary Analysis with eCK-PFS" of the appendix.

Attack Game 4.1 (Mutual authentication and no password reuse). Let attack game G in model eCK-PFS be played by a PPT adversary $\mathcal{A}$ under the assumption that all parties in $\mathcal{P}$ implement mutual authentication and there is no password reuse in the key derivation of device authentication codes or password hashes.

1. $\mathcal{A}$ issues corrupt($\hat{C}_1$) for a $\hat{C}_1 \in \mathcal{P}$. It reveals $PWD_{\hat{C}_1}$ and $DAC_{\hat{C}_1}$. Since the assumption is that $\hat{C}_1$ authenticates servers it communicates with, there exists a $dac \in DAC_{\hat{C}_1}$ for which $dac = (k_{dac}, \hat{S})$.
2. $\mathcal{A}$ prevents the `SESSION_REQUEST` of a $\hat{C}_2 \in \mathcal{P}$ from reaching $\hat{S}$, with which it initiated a session s, where $s_{actor} = \hat{C}_2$, $s_{peer} = \hat{S}$ and $s_{role} = \mathcal{I}$. $\mathcal{A}$ computes $esk_{\mathcal{A}} \overset{R}{\leftarrow} \{0, 1, ..., 2^{256} - 1\}$ and $epk_{\mathcal{A}} \leftarrow \text{X25519}(esk_{\mathcal{A}}, 9)$. With the $epk_{\hat{C}_2}$ contained in the `SESSION_REQUEST`, $\mathcal{A}$ determines the session key with $k_s \leftarrow \text{MSB}_{128}(\text{SHA-256}(\text{X25519}(esk_{\mathcal{A}}, epk_{\hat{C}_2})))$.
3. $\mathcal{A}$ uses send($s, \hat{S}, m$) to start the responder session s' to s on behalf of $\hat{S}$, where m is a `SESSION_RESPONSE` frame, containing $epk_{\mathcal{A}}$. The k_{dac} for $\hat{S}$ from the first step is used as key for CCM.
4. $\hat{C}_2$ accepts this message as it is authenticated with the correct k_{dac} for $\hat{S}$. It computes $k_s \leftarrow \text{MSB}_{128}(\text{SHA-256}(\text{X25519}(esk_{\hat{C}_2}, epk_{\mathcal{A}})))$. $\hat{C}_2$ replies with a `SESSION_AUTHENTICATE` frame, which is encapsulated in a `SECURE_WRAPPER` that is encrypted and authenticated with k_s.
5. $\mathcal{A}$ issues send(s, m) where m is a `SESSION_STATUS` indicating successful authentication, which is encapsulated in a `SECURE_WRAPPER` that is encrypted and authenticated with k_s. The session s is completed.
6. The session s is fresh because:
   a) G does not include the query session-key(s).
   b) There is no session-key(s^*) query issued for any session s^* matching s.
   c) G does not include the ephemeral-key(s) query.
   d) For no origin session s' to session s does G include a corrupt(s_{peer}) and ephemeral-key(s') query.
   e) G does not use corrupt(s_{peer}) and there is an origin session s' to s, namely the one created by the adversary.

   $\mathcal{A}$ issues test-session(s). The challenger provides either the real session key or a random session key. $\mathcal{A}$ can determine with certainty which key they have been given, since they know the session key k_s. Thus, b' is selected accordingly and $P(b = b') = 1$. $\mathcal{A}$ wins G with $\text{Adv}_G^{\text{eCK-PFS}}(\lambda) = 1$. The KNXnet/IP Secure unicast protocol is not secure in eCK-PFS, because no negligible function negl(λ) exists such that $\text{Adv}_G^{\text{eCK-PFS}}(\lambda) \leq \text{negl}(\lambda)$. This is evidently the case, because any negl(λ) would have to fulfill definition 3.34 and $\mathrm{Adv}_G^{\text{eCK-PFS}}(\lambda)$ is a constant non-zero value.

In summary, the AKE for unicast communication in KNXnet/IP Secure is proven to be insecure in the eCK-PFS model by Cremers and Feltz [68]. Since the certain success of the adversary in the eCK-PFS model is caused by the use of symmetric keys for the authentication, the unicast protocol would also be considered insecure in weaker security models, such as eCK by LaMacchia et al. [67] or the original Canetti and Krawczyk model [66].

4.1.4. Model Checking of Session FSM

Model checking in this work has two purposes, the comparison of the textual description of KNXnet/IP Secure with its more formalized specification of the session and timer synchronization FSMs as well as checking the conformance of real device with the ISO 22510:2019 standard by comparing the protocol state fuzzing results with the specified session FSM. The former is a topic in this chapter, while the latter is presented in the "Analysis of Certified Devices" chapter. In order to facilitate the model checking, the FSMs from ISO 22510:2019 are translated into the NuXMV syntax. The alphabet, on which the NuXMV models for the session FSM operate, is extended to include categories of invalid frames, so that the same alphabet is reusable for the protocol state fuzzing. This allows a precise comparison between the session FSM and the protocol state fuzzing results. Additionally, the adaptation of CTLs requires only minor modifications.

In this section the model checking is conducted with the session FSM, visualized in figure 19. Two NuXMV models were created from the specification in ISO 22510:2019. The one labeled "Session FSM with Timer" is trying to replicate the specification exactly, including the modeling of time progression and limits for the timer. The other labeled "Session FSM" is a simplified version, that replaces the timer with a state variable, which indicates whether the timer is running or has expired. While the simplified version is not strictly required, it was originally created to reduce the complexity of the model in order to avoid a state-space explosion and permit a faster evaluation in NuXMV. Details about both models can be found in section "A.7 Software, Models and Logs" of the appendix.

Test conditions for the model checking are derived from the requirements for the server's unicast implementation in ISO 22510:2019. The selection is based on the relevancy of the requirements and the precision of their wording. The latter is necessary to faithfully translate them to CTLs, because otherwise the required amount of interpretation and context information would call into question whether the paragraphs in ISO 22510:2019 are accurately represented by them. The model checking is conducted with 23 tests, as shown in tables 3 and 4. The CTLs for every test are listed in table 15 of the appendix.

The majority of the tests are successful and the CTLs are proven to be satisfied by NuXMV, which means that the description in ISO 22510:2019 and the session FSM do not contradict each other. All states in the session FSM can be reached and all transitions can be activated, which is proven in tests 21 and 23 respectively. However, there are five cases that require further inspection.

The test 5 is about the termination of the AKE when the client fails to respond to the server's `SESSION_RESPONSE` within 10 seconds. The session FSM does provide a transition from the `UNAUTHENTICATED` state to the `IDLE` state when the timer expires. However, the session FSM as specified, is non-deterministic. This is proven in test 22. The timer can simultaneously expire as another event occurs, thus two transitions would in theory be able to fire. This is an edge case and presumably the specification intends the timer expiration to take precedence. However, checking if the timer reaches zero in `UNAUTHENTICATED` always leads to the `IDLE` state, is not fulfilled due to non-determinism. This has no real world consequences, because implementations would be deterministic, since one of the possible options has to be picked. The session FSM issue with non-determinism is thus rather pedantic albeit accurate.

Model	**Test No.**											
	1	2	3	4	5	6	7	8	9	10	11	12
Session FSM	✓	✓	✓	–	–	×	✓	✓	✓	✓	✓	✓
Session FSM with Timer	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓

Table 3: Model checking of the session FSM in NuXMV - Part 1
Symbols: ✓fulfilled, × unfulfilled; – not applicable
See table 15 for test details.

Model	**Test No.**										
	13	14	15	16	17	18	19	20	21	22	23
Session FSM	–	○	✓	✓	✓	✓	×	✓	✓	×	✓
Session FSM with Timer	–	○	✓	✓	✓	✓	×	✓	✓	×	✓

Table 4: Model checking of the session FSM in NuXMV - Part 2
Symbols: ✓fulfilled, ○ ambiguous, × unfulfilled; – not applicable
See table 15 for test details.

Test 6 evaluates the veracity of the following statement: "If the client sends any other `SECURE_WRAPPER` except a wrapped `SESSION_AUTHENTICATE` request before successful authentication, the server shall respond with a `SESSION_STATUS` with status field of `STATUS_UNAUTHENTICATED`" [6, p. 144]. This is evidently not true, when comparing it with the session FSM. When the server is in the `UNAUTHENTICATED` state, the client could send a `SECURE_WRAPPER` containing a `SESSION_STATUS` frame with the status field set to `STATUS_CLOSE`. This would trigger event `E03`, which means that the server performs the action `A3`. Thus the servers sends a `SECURE_WRAPPER` containing a `SESSION_STATUS` frame with the status field set to `STATUS_CLOSE` back to the client. Another counterexample is that the client could send a `SECURE_WRAPPER` containing a

SESSION_STATUS frame with the status field set to STATUS_AUTHENTICATION_SUCCESS. This would not happen with a client that conforms with the standard, but according to the session FSM no transition from the UNAUTHENTICATED state would be triggered. The quote from ISO 22510:2019 is therefore inaccurate and would have to be corrected to match the session FSM.

According to ISO 22510:2019 the server is supposed to close the TCP connection when the KNXnet/IP header is not well-formed [6, p. 41]. This is evaluated in test 14. The specification of the session FSM does not mention this case. Thus, it does not satisfy the test condition, as the reception of a malformed header in the UNAUTHENTICATED state does not result in the immediate termination of the session or TCP connection. However, it could be argued that the header should be validated before being processed by the session and thus is not its responsibility. Therefore, the test result is considered ambiguous, although the CTL is not fulfilled.

The standard states that when a SESSION_AUTHENTICATE frame has the reserved field set to a non-zero value, it shall be discarded [6, p. 129]. In test 19, it is determined that this does not match the specification of the session FSM. In other instances when ISO 22510:2019 refers to discarding a frame, it means that the frame is ignored without any further action. However, if the server receives a valid SECURE_WRAPPER with an invalid SESSION_AUTHENTICATE frame in the UNAUTHENTICATED state, then this is event E02 which triggers action A2. The server is supposed to reply with a SECURE_WRAPPER containing a SESSION_STATUS with the status field set to STATUS_AUTHENTICATION_-FAILED. This contradicts the statement that the frame shall be discarded. It is unclear which behavior is the intended one.

In summary, the description of the server's behavior for KNXnet/IP Secure unicast communication mostly matches the session FSM specified in ISO 22510:2019. When differences occurred, they were caused by oversimplifications or ambiguities in the text, or edge cases that were not considered in the FSM. None of the identified issues have an impact on the session's security.

4.1.5. Improvement Suggestions

The core problem of the KNXnet/IP Secure unicast protocol is that the KNX Association decided to develop their own AKE, while disregarding scientific publications about AKE protocols with security proofs and not fully addressing the criticism of their early design draft by Judmayer et al. [1]. This resulted in a design with an insecure authentication, that relies on symmetric keys shared among communication partners. Additionally, the keys are derived from passwords with PBKDF2 using a static salt for clients and servers respectively.

There are two general recommendation for the development process and future versions of KNXnet/IP Secure unicast. Firstly, an AKE protocol from the literature with suitable security properties should be integrated, instead of the KNX Association developing their own. Despite being aware of the STS protocol by Diffie et al. [48], which would have been a good choice, the KNX Association decided not to adopt it. Secondly, criticism by security researchers should be taken serious. Several of the conceptual flaws that still exist in ISO 22510:2019 were pointed out by Judmayer et al. in their publication from 2014. The submission of the finalized KNXnet/IP Secure design to the International Organization for Standardization (ISO) happened in 2017 [20]. This should have been ample time to address all design problems, but the KNX Association chose not to. It also seems that the protocol design ideas by Granzer et al. [25] were not used in KNXnet/IP Secure.

A specific suggestion for the next KNXnet/IP Secure version is to adopt asymmetric cryptography for the authentication by integrating an AKE based on the STS protocol [48] with pre-shared public keys and CCM for AEAD. Considering the application on embedded hardware, Ed25519 [136] could be a suitable choice for the signatures. During the initial configuration, each server would generate a key pair as a replacement of the device authentication code. The ETS would be given the public keys while the private keys do not leave the devices. Similarly, the ETS can generate key pairs for the management users, replacing the password hashes. Other clients would generate key pairs for their users too and provide the public keys to the ETS. In the next step the ETS would distribute the public keys of the servers to potential communication partners. Each server would also receive the public keys for its users. Pre-sharing the public keys avoids the inclusion of a public key infrastructure (PKI) and certificates into the protocol. The specification of the `SESSION_RESPONSE` and `SESSION_AUTHENTICATE` would have to be modified. Both need to include a signature of the concatenated public keys X and Y [48, p. 9], not $X \oplus Y$ as specified in ISO 22510:2019 for KNXnet/IP Secure. Additionally, the signature in `SESSION_RESPONSE` would have to be encrypted with the session key.

The benefit of this approach is that a compromise of a device does not reveal secret information about its communication partners anymore, preventing their impersonation. It eliminates the risk of online and offline attacks on the authentication, under the assumption that the ECDLP is intractable. The adapted STS with encryption would also satisfy all relevant security properties, which excludes key control.

At a bare minimum, the KNXnet/IP Secure protocol should not use static salts, replace PBKDF2 with a modern key derivation function such as Argon2id [131], and enforce through the ETS that strong passwords are used. Additionally, the factory default management passwords for the server should not be a static key. Instead, a randomly generated per device value like the FDSK has to be assigned. However, as these changes would already require an entirely new version for KNXnet/IP Secure due to incompatibility with the current one and do not address the core problem, the AKE has to be fixed properly as previously described.

It would also be advisable to address the ambiguities, unspecified parts and errors that have been identified. This includes adding the missing parts of the formatting function originally taken from KNXnet/Data Secure, properly specifying the client-side, preventing nonce reuse due to session number overflow, and correcting the length and HPAI validation of `SESSION_REQUEST` frames. Furthermore, the quality assurance for documents like the ISO 22510:2019 has to be improved. For instance, figures 73, 74, 102 and 103 do not contain the correct images, they have been replaced with already existing ones from the document.

4.2. Multicast

The security properties of the multicast protocol are analyzed in a similar fashion to the unicast protocol. The application of CCM, with its custom formatting and block generation function, is checked for the fulfillment of definitions 3.3 and 3.2 to identify potential problems that may affect the confidentiality or authentication. Furthermore, the group communication is examined with regard to the claimed security properties of confidentiality, mutual authentication, data integrity and resistance against replay attacks, as previously highlighted in “3.3 KNXnet/IP Secure in ISO 22510:2019”. The evaluation also determines whether flaws found by Judmayer et al. [1] in drafts of KNXnet/IP Secure have been addressed in ISO 22510:2019. Furthermore, it is checked if the access control solves design flaws of KNXnet/IP, discussed in subsection “3.1.6 Insecurities and Design Flaws in KNXnet/IP”. KNXnet/IP Secure’s multicast does not implement an AKE, hence no formal analysis is conducted with eCK-PFS. Model checking in NuXMV is applied to identify possible mismatches between the timer synchronization FSM, depicted in figure 22, and the described behavior in ISO 22510:2019, as well as potential errors in the model. Improvement suggestions for the multicast protocol of KNXnet/IP Secure are made, based on the results of all previous steps of the analysis.

4.2.1. CCM Requirements

ISO 22510:2019 implicitly defines the same formatting and counter generation function for both unicast and multicast, shown in figures 11 and 12. This has been previously explained in “3.3.1 CCM”. The requirements from NIST SP 800-38C, as shown in definition 3.3, have already been evaluated for unicast in “4.1.1 CCM Requirements”. Since the first two requirements are only based on the definition of the formatting function, which both protocols share, they are equally fulfilled for multicast. According to the third requirement B_0 has to be different from all counter blocks Ctr_i for all invocations of CCM under the same key. Multicast applies CCM to the `SECURE_WRAPPER` and `TIMER_NOTIFY` frames. `TIMER_NOTIFY` frames have a fixed length, requiring only one counter block for CCM. The B_0 is not equal to Ctr_0, because $\mathrm{LSB}_{16}(B_0) \neq \mathrm{LSB}_{16}(Ctr_0)$, as shown in definition 3.43. Since the `SECURE_WRAPPER`’s B_0, Ctr_i, and length limits for the payload are identical to unicast, the B_0 is different to all Ctr_i, as explained in “4.1.1 CCM Requirements”. Hence, the third requirement is fulfilled.

Lastly, it has to be checked in accordance with definition 3.2, that the nonce N is never repeated under the same key for different input data. The backbone key, that is used for CCM by all multicast group members, has an unlimited lifetime [6, p. 111]. Thus, it can be assumed that the key is the same for all invocations of CCM in multicast for the purpose of this analysis.

The nonce N for `SECURE_WRAPPER` frames in multicast consists of the timer value, serial number and message tag, because $N = \mathrm{MSB}_{112}(B_0)$, as explained in “3.3.1 CCM”. The inclusion of the serial number and message tag in the nonce is a deliberate choice to ensure different Ctr_i, even when the timer value is identical [6, p. 108]. This rectifies a design flaw in AN 159/13 v02 discovered by Judmayer et al., where nonce reuse would occur when frames were sent with the same timer value [1, p. 5-6]. In ISO 22510:2019 the nonces should be different even when two or more devices construct their frames when their timer values are identical, because the serial number should be unique for every device in the KNX installation and the 16 bit message tag is supposed to be randomly chosen for every frame. If one device constructs two frames with the same timer value, then there is a chance of $1/2^{16}$ that the same message tag is picked for both frames, which would cause a nonce reuse. From a theoretical standpoint this still poses a significant risk to the confidentiality. This is a realistic problem, if the implementation is able to construct frames faster than the update frequency of the timer, which is "[...] one tick per millisecond real-time" [6, p. 114]. This risk could be eliminated by requiring that a timer increase has to have taken place before constructing the next frame, similar to the sequence numbers in unicast.

ISO 22510:2019 assumes that the multicast timer never overflows [6, p. 111]. It is unclear how devices are supposed to handle the case when the timer reaches its maximum as they cannot further monotonically increase it every millisecond and never decrement it. This is a similar issue in the specification as with the sequence numbers in the unicast protocol. There are two possible options, either the timer wraps around to zero or it stays at the maximum value. In the former case, it does not necessarily cause a nonce reuse, since the specific timer might not have been used in a frame before and the message tag still contributes 16 random bit for each frame. However, if the timer is stuck at the maximum and is not reset by changing the backbone key, then a nonce reuse is expected. Since all subsequent frames use the same timer value, only the message tag varies. However, due to its short length there are only $2^{16} = 65536$ possible values. Related to the birthday paradox, the probability of finding two frames with equal message tags out of n random frames is:

$$P(n) = 1 - \frac{2^{16}!}{(2^{16} - n)! \times 2^{16n}} \tag{44}$$

The probability is larger than 50% for $n = 302$. Therefore, after 302 frames sent by one device it is probable that nonce reuse can be seen, which breaks the confidentiality.

The KNX Association dismisses the risk of the timer reaching the maximum value in ISO 22510:2019, stating that with "[...] timer ticks every millisecond, an overflow of the timer would theoretically occur after 9 thousand years" [6, p. 111]. The standard requires that devices that cannot immediately persist their timer during power-off have to store it in regular intervals and to ensure that the timer does not run backwards, they have to increment it by one interval on power-up [6, p. 133]. ISO 22510:2019 limits the persistence interval to a maximum of one hour [6, p. 133]. It thereby tries to prevent an adversary with physical access from pushing the timer to its maximum by continuously rebooting the device, as discussed in the appendix of AN 159/13 v06 [49, p. 78]. Since the maximum increase of the timer per reboot is $1\,\text{h} = 3.6 \times 10^6\,\text{ms}$, it takes at least $2^{48}/3.6 \times 10^6 \approx 7.819 \times 10^7$ reboots to reach the maximum. If each reboot takes 1 s, then it would require approximately 905 days. This might not be a practical attack, because the boot time could be significantly higher, the persistence interval shorter or the devices might not use persistence intervals, because they are able to store the timer on power-off. However, it should not be assumed that a multicast timer cannot overflow. In theory there could be bugs in the software or malfunctioning hardware that result in wrong timer values. Additionally, specialized attacks against specific hardware or software could exist, that permit to influence the timer. Therefore, it would still be important to specify how devices have to handle the timer overflow, to avoid breaking the confidentiality of past, current and future communication of the multicast group. From a theoretical standpoint the requirement of definition 3.2 is not fulfilled.

Contrary to the `SECURE_WRAPPER`, nonce reuse for `TIMER_NOTIFY` frames is not relevant, since when the timer value and message tag are identical in two frames, then they are the same frame.

4.2.2. Group Communication

The multicast protocol encrypts and authenticates the communication among group members with the backbone key. As explained in "3.3.3 Multicast", this is a single key with unlimited lifetime that is known to all group members. The compromise of a single group member and its backbone key allows an adversary to decrypt all past and future communication of the group and impersonate group members. This obvious flaw already existed in AN 159/13 v02 and was pointed out by Judmayer et al. [1, pp. 4-5]. It is also not possible to authenticate individual group members, since the backbone key is the only key material used in the protocol.

In order to protect against replay attacks, every device has a multicast timer that the group attempts to keep synchronized through the process described in "3.3.3.5 Timer Synchronization Finite State Machine". However, frames are only rejected if the contained timer value is more than `PID_MULTICAST_LATENCY_TOLERANCE` behind the local timer. As ISO 22510:2019 states, this means that "frames with slightly past timer values shall be accepted to account for network latency" [6, p. 111]. Hence, any frame containing a timer value within this time span can be replayed to the respective target device. `PID_MULTICAST_LATENCY_TOLERANCE` has a default value of 2 seconds [6, p. 113]. The same issue already existed in AN 159/13 v02 according to Judmayer et al. [1, p. 5]. They also claimed that a replay attack would be possible after a device is booted [1, p. 5]. This risk is acknowledged by ISO 22510:2019, any `SECURE_WRAPPER` or `TIMER_NOTIFY` with a timer more recent than the device's unsynchronized timer, could be replayed [6, p. 117]. The standard specifies an optional procedure to acquire an authentic timer before accepting any frame, which was previously described in "3.3.3.5 Timer Synchronization Finite State Machine". While no flaw could be identified in the described steps, assuming that there is at least one other legitimate group member that replies, it is not mandatory and as such standard conforming devices could still be susceptible to the replay attack.

Another risk for replay attacks occurs should the multicast timer reaches its maximum. This was again first identified in [1, p. 5] and later considered in the appendix of AN 159/13 v06 for when a warp around of the timer occurs [6, p. 78]. While the KNX Association does not appear to consider this case likely to happen, it would permit to replay all past frames with a higher timer value. Another interesting case is when the timer would be stuck at the maximum. Assuming that the timer is synchronized among group member, each of them would accept frames that contain a timer value that is not more than `PID_MULTICAST_LATENCY_TOLERANCE` before the maximum timer value. Thus, every frame that is sent after the timer reached its maximum can be replayed indefinitely. While preventing replay attacks is one of the goals of the multicast protocol, the standard concedes that Data Secure with device specific counters is required to minimize the risk [6, p. 111]. The fairly complex timer synchronization, depicted in figure 22, does not sufficiently solve the problem.

The access control for the routing service family is configured through `PID_SECURED_-SERVICE_FAMILIES`, as previously explained in "3.3.3.6 Access Control". The standard requires that the routing service may not be accessed through plain KNXnet/IP frames when secure communication is enforced. Thus, it addresses the security concerns that were raised for KNXnet/IP with regard to routing in "3.1.6 Insecurities and Design Flaws in KNXnet/IP". The secrecy of the backbone key should prevent an adversary from issuing requests that would pass authentication. However, the aforementioned design flaws of the multicast protocol could allow a circumvention of the access control, for instance through replay attacks.

According to ISO 22510:2019 the remote diagnosis and configuration service is not supported when the security mode is enabled [6, p. 110]. This mode controls whether certain service may only be accessed through Data Secure [122, p. 56]. It is not clearly stated under which circumstances the security mode is required to be activated in relation to KNXnet/IP Secure. ISO 22510:2019 specifies that properties, such as the backbone key, password hashes and device authentication code, shall only be writable through the application layer, secured with the tool key [6, pp. 133-135]. This means Data Secure is mandatory for their configuration, but this appears to be generally enforced and unrelated to the security mode. It seems that the security mode is supposed to get enabled when a system broadcast is used to configure the routing multicast address and backbone key through KNXnet/Data Secure [6, p. 88]. However, the relevant sentence in ISO 22510:2019 is incomplete, missing the crucial word: "In addition, device security mode shall be." [6, p. 88]. Therefore, it is not well-defined if the security mode gets enabled when a device has been configured to use KNXnet/IP Secure. Thus, considering ISO 22510:2019 as written, it would be possible that the device has its KNXnet/IP Secure properties configured through KNXnet/Data Secure, but the security mode is disabled. Presumably this is not intended, as access to the remote diagnosis and configuration service would not be limited, thus permitting unauthorized configuration of the device. Due to the ambiguous and incomplete specification the concerns raised in "3.1.6 Insecurities and Design Flaws in KNXnet/IP" about access control are not fully addressed. In practice KNXnet/IP Secure devices appear to activate the security mode when the ETS configures them with the "secure commissioning" option enabled.

The discovery service does not require authorization [6, p. 109]. The implications were already discussed in “4.1.2 Authenticated Key Exchange and Session” and equally apply to multicast.

The evaluation of the security properties defined in “3.2.4 Security Properties for Cryptographic Protocols” had the following results:

- **Confidentiality** is not always ensured, due to the possibility of nonce reuse in the multicast protocol. Hence, this goal of ISO 22510:2019 is not generally fulfilled.
- **Data integrity** is ensured through CCM for AEAD with the backbone key. This goal of ISO 22510:2019 is fulfilled.
- **Message authentication** is not ensured, as the no particular member of the group can be authenticated as the origin of a frame, since every member uses the same key. The protocol can only authenticate that the group is the origin of the frame.
- **Entity authentication** is not ensured, as message authentication for individual members is not provided.
- **Mutual authentication** is not ensured, as entity authentication is not provided. ISO 22510:2019 stated the “[…] goal of mutual authentication of the nodes involved in KNXnet/IP multicast communication […] is to establish a trusted group of devices communicating together” [6, p. 102]. Evidently the protocol design only permits to prove knowledge of the backbone key and to this extent membership of the group. However, since it is not possible to authenticate that a particular members has sent a frame, mutual authentication as defined in cryptography cannot be accomplished.
- **Non-repudiation** is not fulfilled, because a group member can plausibly deny that they have sent a particular frame, since any other group member could have created an identical frame, as they also know the backbone key.
- **Resistance against replay attacks** is not ensured, as previously shown. Replay attacks are always possible within a limited time span and when the multicast timer reaches its maximum. This goal of ISO 22510:2019 is not fulfilled.
- **Perfect forward secrecy** is not ensured, because a compromise of the backbone key allows to decrypt all past frames of the multicast group.
- **Resistance to known session key attack** is not applicable, because there is no session or AKE. If the backbone key is treated as a session key then its compromise breaks both confidentiality and authentication.
- **Resistance against key compromise impersonation** is not ensured, because a compromise of the backbone key allows the adversary to pose as a group member to any legitimate device in the group.
- **Resistance against unknown key-share attack** is not applicable, because there is no AKE.

- **Implicit key authentication** is not fulfilled, because no individual group members can be authenticated.
- **Key confirmation** is provided, since the shared knowledge of the backbone key permits to verify that a frame has been encrypted and authenticated with the correct key.
- **Explicit key authentication** is not fulfilled, because implicit key authentication is not provided.
- **Key freshness** is not fulfilled, because the backbone key has an unlimited lifetime. ISO 22510:2019 suggests a manual change of the backbone key when a compromise has been detected [6, p. 111]. This does not solve the problem in the protocol design though.
- **Key control** is not applicable, as there is no AKE.
- **Identity hiding** is not applicable, as there is no AKE.

In summary, the multicast communication of KNXnet/IP Secure fails to fulfill a majority of its goals stated in ISO 22510:2019. It does not fully protect against replay attacks or provide mutual authentication. Confidentiality could be broken due to potential nonce reuse. The only fulfilled goal is that data integrity is provided. KNXnet/IP Secure multicast cannot be considered a secure protocol based on the current state of the art in cryptography.

4.2.3. Model Checking of Timer Synchronization FSM

The model checking is continued with the timer synchronization FSM, depicted in figure 22. The goal is to identify potential errors, ambiguities and deviations between the description in the text of ISO 22510:2019 and the FSM model. For this purpose two NuXMV models were created from the specified FSM. The first model is labeled "Timer Sync FSM Integer Clock". It tries to represent the specified FSM as closely as possible, while modeling the time progression and multicast timer value through integer ranges. This has the benefit that CTLs can be used to express properties and prove whether they are true through BDD. However, the time range that can be effectively evaluated is limited, because of the state-space explosion that would otherwise occur. Thus, the time range has to be restricted. The second model is labeled "Timer Sync FSM Real Clock". It uses NuXMV's Real data type that permits unbounded variables that store rational numbers, contrary to its name. It can accurately express the timer synchronization FSM, but NuXMV does not permit the evaluation of CTLs on models that use the Real data type. Thus, LTLs with BMC have to be used. As previously explained in "3.4 Model Checking", BMC can generally only prove that a condition is not fulfilled. The two models are therefore complementing each others by addressing their respective shortcomings.

Test conditions for the model checking are derived from the description of the timer synchronization process in ISO 22510:2019. The selection of statements from the standard is based on their relevancy and precision. The latter is required to translate them into CTLs and LTLs. It should be noted nearly all CTLs and LTLs had to be modified to account for the non-determinism of the timer synchronization FSM, as otherwise they would not have been fulfilled. This is an issue, because it is not well-specified that only one of the events occurs at a time, as they a defined through conditions that have to be met, which are based on external inputs and the current values of internal state variables. Thus, there are cases where the requirements for multiple events are fulfilled simultaneously and it is unclear which one is triggered. Amending the CTLs and LTLs when necessary is the pragmatic solution to analyze the timer synchronization FSM, with the knowledge that implementations would be deterministic. The model checking is conducted with 16 tests, as shown in table 5 and 6. The respective CTLs and LTLs are listed in table 16 of the appendix.

Model	**Test No.**											
	1	2	3	4	5	6	7	8	9	10	11	12
Timer Sync FSM Integer Clock	✓	✓	×	✓	×	✓	×	×	✓	–	✓	✓
Timer Sync FSM Real Clock	○	○	×	○	×	○	×	×	○	×	○	○

Table 5: Model checking of the timer synchronization FSM in NuXMV - Part 1
Symbols: ✓fulfilled, ○ not refuted, × unfulfilled; – not applicable
See table 16 for test details.

Model	**Test No.**			
	13	14	15	16
Timer Sync FSM Integer Clock	✓	✓	×	✓
Timer Sync FSM Real Clock	○	–	–	–

Table 6: Model checking of the timer synchronization FSM in NuXMV - Part 2
Symbols: ✓fulfilled, ○ not refuted, × unfulfilled; – not applicable
See table 16 for test details.

The majority of the tests are successful, with 10 out of 16 CTLs proven to be satisfied by NuXMV. All states in the timer synchronization FSM can be reached and all transition can be activated, which is verified with tests 14 and 16. However, there are 6 cases that require further inspection.

Test 3 evaluated the statement: "The device timer shall not be decreased in any case" [6, p. 112]. This is evidently not true, when considering the case where timer wraps around to zero after reaching the maximum. While excluded in this test, changing the backbone key would also cause a reset to zero. While other parts of the specification describe these cases, this specific statement overgeneralizes.

Test 5 checks the claim that "A `TIMER_NOTIFY` shall be sent by any device if no recent-enough `TIMER_NOTIFY` or `SECURE_WRAPPER` frame has been received for about 10 s" [6, p. 112]. This statement could either be true or false depending on how lenient the interpretation is. Initially, the current state could be `SCHED_PERIODIC` and a

TIMER_NOTIFY with a timer value larger than the multicast timer has been received, thus event E01 with actions A1 + A9 + A3 is triggered causing the notify_timer to be potentially set to 12.8 s, which is the maxDelayTimeFollowerPeriodicNotify. The FSM remains in SCHED_PERIODIC and the period mentioned in the quoted statement starts. After nearly 12.8 s have passed a not recent enough TIMER_NOTIFY is received. This causes E04 with A4 to be triggered, which possibly resets the notify_timer to a maximum of 2.7 s, which is the maxDelayTimeFollowerUpdateNotify. The FSM transitions into the SCHED_UPDATE state and the notify_timer would then have to expire first, before a TIMER_NOTIFY is sent. It should be noted though, that this is the worst case and not the average case. Furthermore, the CTL and LTL test for the sending of the TIMER_NOTIFY to occur at the very next step in the FSM, since the semantics of "about 10 s" [6, p. 112] are rather vague. Thus, a TIMER_NOTIFY will be sent eventually, but it may take more than 10 s of not receiving a recent enough frame. Given that the FSM operates with time spans measured in milliseconds and seconds, the 12.8 s were considered too big of difference compared to the quoted statement.

Test 7 evaluates the statement: "Any device receiving a valid outdated TIMER_-NOTIFY or SECURE_WRAPPER frame shall schedule sending a TIMER_NOTIFY frame after a random delay of usually up to around 3 s (maxDelayTimeFollowerUpdateNotify)" [6, p. 112]. It is technically incorrect, because if the FSM is already in the SCHED_UPDATE state, then receiving a valid outdated frame will not restart the schedule of sending a TIMER_NOTIFY. It would have no effect as this is either event E04 or E08. However, the statement would be correct if it specified that this applies in state SCHED_PERIODIC.

Test 8 evaluates the claim: "If a TIMER_NOTIFY or SECURE_WRAPPER frame with a timer value greater than the sync latency tolerance (fraction of the overall latency tolerance) behind the local timer value as time stamp field is received before the delay elapses, the schedule shall be cancelled" [6, p. 112]. This statement is incorrect, as when the received timer value is more than the syncLatencyTolerance behind the multicast timer, then either event E03 with A0 or E07 with A2 is triggered in both SCHED_UPDATE and SCHED_PERIODIC, as shown in the FSM in figure 22. The schedule is not canceled in either case.

Test 10 checks if the following statement is true: "Devices that claim to be a time keeper for their secure KNXnet/IP multicast group continue scheduling TIMER_NOTIFY frames on outdated received TIMER_NOTIFY or SECURE_WRAPPER frames in a shorter and earlier time window than the other devices" [6, p. 112]. This statement is technically incorrect, because when current state is SCHED_UPDATE, then the reception of outdated frames would not trigger the scheduling as there is already a schedule running, see E04 and E08 in figure 22. The non-determinism is proven in test 15. The events E01 to E08 are correctly specified though, as they are mutually exclusive.

In summary, the description of the timer synchronization for KNXnet/IP Secure multicast mostly matches the FSM specified in ISO 22510:2019. A majority of the test failures stem from pedantic interpretation or vague descriptions in the text of ISO 22510:2019 that could be fixed with minor modifications. Only one test identified a factual error, which is test 8. None of the findings have a notable impact on the security of the multicast protocol.

4.2.4. Improvement Suggestions

Due to the identified security flaws it would be advisable to redesign the multicast protocol. Secure GKE protocols are an ongoing topic of research. A core challenge is the scalability with the number of group members and limited hardware resources, while achieving desirable security properties such as forward secrecy and post-compromise security. The former ensures that past sessions are not affected by a later compromise, while the latter ensures that future sessions provide security guarantees despite a current compromise, see Cohn-Gordon et al. [137] for details.

The section "2.7 Group Key Exchange" in the "Related Work" chapter already introduced different GKE protocols that could be applied in KNX. This includes older protocol designs such as Burmester and Desmedt [55], CLIQUES by Steiner et al. [56], TGDH by Kim et al. [57], [58] and the extended STR by Kim et al. [54]. However, CLIQUES, TGDH and extended STR do not include authentication in their design [56, p. 18], [58, p. 66], [54, p. 233], an authenticated communication channel needs to be provided by the surrounding protocol in which they are applied. As Mark Manulis points out in [59], this could be accomplished through digital signatures, certificates and a PKI. In theory, it would also be possible to pre-share public keys, but this would require a static group and enough memory to store them on every device. While the multicast group membership in KNXnet/IP Secure is not dynamic, the available memory of the embedded devices would likely not be enough to store all keys for the maximum number of KNXnet/IP Secure devices that could potentially be part of the multicast group. Therefore, a PKI would likely be needed. More recent protocols that implement CGKA are TreeKEM [61] and TTKEM [63], which are supposed to provide both forward secrecy and post-compromise security. These protocols require a PKI [61, p. 9], [62, p. 10], which would need a dedicated device in the architecture of the KNX installation.

While existing GKE and CGKA protocols might not be a perfect fit for KNXnet/IP Secure and a trade-off between available hardware resources and security may be necessary, it is evident that solutions exist which would provide significant improvements to the security compared to the current multicast protocol. Additionally, nonce reuse should be prevented by disallowing identical timer values in subsequent frames sent from the same device and not permitting the timer value to be stuck at the maximum while continuing group communication. Ideally, the timer synchronization would be replaced with a more robust method to prevent replay attacks. Furthermore, the standard should properly explain the interaction between KNXnet/IP Secure and Data Secure.

5. Device Management with the ETS5

The device management is conducted through the ETS software. This chapter investigates how the passwords for the AKE of KNXnet/IP Secure are assigned by the ETS5 v5.7.6 and under which circumstances the identified offline attack against the authentication is practical. Furthermore, the ETS5 v5.7.6 has to store the symmetric keys and passwords for the KNX installations that it manages. Thus, it is of interest to check whether they are stored securely. This also applies to exported projects, which contain the cryptographic secrets as well. The update process for the ETS5 is analyzed to ensures that an adversary cannot supply a malicious executable. An evaluation of the conformance with the ISO 22510:2019 standard is conducted through black-box tests. Lastly, improvements suggestions for future versions of the ETS are provided, based on the findings.

5.1. Device Commissioning

The ETS5 provides a product database from which devices can be added to the project. When the first KNXnet/IP Secure device is included, the ETS5 requires the user to choose a project password. The UI claims, that it would be used to protect the stored keys, as can be seen in figure 29 in the appendix. All password input fields allow only characters from the printable ASCII range, see RFC 20 [138], and the length is limited to a maximum of 20 characters. Additionally, the ETS5 displays a password strength based on the estimated entropy provided by the input. The calculation considers the length and whether uppercase, lowercase, numbers and special characters are included. For instance, a password of length eight, that contains characters from all the categories, is considered “good” by the ETS5.

With every added KNXnet/IP Secure device the ETS5 requests the device certificate to be entered, which encodes the device’s FDSK and serial number in a QR code and string. Typically, it is printed on a sticker placed on the device. The option for “Secure Commissioning” is enabled by default and activates the use of KNXnet/IP Secure and Data Secure for the configuration. The ETS5 automatically generates a tool key for KNXnet/Data Secure. Once the device is added to the project, the user can adjust device specific settings in the project. If the real device is connected to the local network, its configuration and application can be modified by the ETS5 to match the state of the project file. The UI refers to this process as a “download” onto the target device. When a KNXnet/IP Secure device is configured for the first time, the ETS5 automatically generates a management user password and device authentication code. It also creates the backbone key when the first KNXnet/IP Secure router is added to the project. Furthermore, if the device supports tunneling and the “Secure Tunneling” option is enabled, it will also create a user password for each supported tunnel. The UI refers to the management user password as the “Commissioning Password” and the password from which the device authentication code is derived as the “Authentication Code”. The tool and backbone key are properly created with a cryptographically secure pseudorandom number generator (CSPRNG). All generated passwords have 8 characters that are randomly chosen from the 95 printable ASCII characters. The passwords can be changed by the user. There does not appear to be an option to reset or regenerate the backbone key in the ETS5. This is contrary to the suggestion in ISO 22510:2019, that the backbone key could be changed when a compromise of a group member is noticed, in order to restore security for the group [6, p. 111].

5.2. Offline Attack against Authentication

An offline attack against the authentication, which may permit an adversary to determine the device authentication code, was previously described in “4.1.2 Authenticated Key Exchange and Session”. A custom module for the GPU-accelerated password cracking software hashcat [133] was written to evaluate the difficulty of brute-force and dictionary attacks against KNXnet/IP Secure’s authentication of the server. The implementation adopts optimization techniques by Choi and Seo [72] and Visconti et al. [70] for the HMAC computation. It was found that the optimization techniques for SHA-256 suggested by Choi and Seo did not result in a measurable performance improvement, because the compiler already made similar optimizations. The implementation has been contributed to the hashcat project and is included in official releases since v6.2.0.

In order to determine the efficiency of brute-force attacks against the AKE, measurements on real hardware are required that provide an estimate on the time needed to complete an exhaustive search. The experiment was conducted on a NVIDIA GeForce GTX 1080 Ti with passwords lengths from 1 to 10. The character set is assumed to consist of the 95 printable ASCII characters. For each password length, ten measurements of the hash rate are made, each after approximately one minute of the brute-force attack running, to account for possible fluctuations. In this context the hash rate refers to the number of passwords tested per second. The required time can be estimated based on the number of possible passwords of length x, which is 95^x, divided by the median hash rate. The results are shown in table 7.

Device authentication codes based on passwords up to 6 characters can be checked exhaustively on a single GPU in little over one year. Therefore, those passwords can be practically attacked through brute-force. However, by default the ETS5 generates password containing 8 characters. It is not practical to brute-force those passwords on a single GPU. The question is whether it is realistically possible to build a GPU cluster that could brute-force 8 character passwords. It would require 9371 NVIDIA GeForce GTX 1080 Ti to iterate through all 8 character passwords within a year. The power consumption of a single unit is supposed to be at most 250 Watt [139]. This would mean a power consumption of 20.522 GWh per year. For comparison, a steel mill with an electric arc furnace could have a power consumption of 572 GWh per year [140]. According to the monitoring report of the Bundesnetzagentur from 2020, the average price without taxes for customers in the industry with a consumption of 24 GWh were 16.54 ct/kWh [141]. This would mean a cost of 3.394 million euros for power supply of the GPUs. When the GPU model was released, the price was 820 euro per unit [142]. This would incur a cost of about 7.7 million euros. While this rough estimation of costs and energy consumption does not include the building, cooling, other hardware, labor costs and taxes, they are within an order of magnitude that could be realistically achieved. While it would be highly uneconomical to construct and operate such as GPU cluster to determine a single device authentication code, a well-designed cryptographic protocol should have made it physically infeasible to gain the required computational power to perform a brute-force attack within a year.

Thus, the default of 8 character passwords is too short, especially considering that future generations of GPUs and ASICs will improve further upon the efficiency and computational power. A default password length upwards of 10 characters should be considered by the KNX Association or ideally a protocol design that does not rely on passwords.

Performing a dictionary attack against KNXnet/IP Secure is much more efficient. For instance, testing a list of 10 million common passwords would require less than 8 minutes on a single NVIDIA GeForce GTX 1080 Ti. If the KNX installation does not use randomly generated, but user supplied passwords, this is a significant risk.

Password Length	**Median Hash Rate**	**Approx. Required Time**
1	(inaccurate) 1.0 H/s	1 min 6 s
2	150.0 H/s	1 min
3	6591.5 H/s	2 min 10 s
4	23238.5 H/s	58 min 25 s
5	22088.0 H/s	4 day 1 h
6	22980.0 H/s	370 days
7	22710.0 H/s	35591 days
8	22434.0 H/s	9371 years
9	22220.5 H/s	898801 years
10*	22220.5 H/s	8.5386100×10^7 years

Table 7: Brute-force attack against device authentication code
Measurements are conducted on a NVIDIA GeForce GTX 1080 Ti with hashcat 6.2.4
The measurements are rounded
*Could not be executed, extrapolation based on previous hash rate

5.3. Insecure Storage of Cryptographic Secrets

The ETS5 has to store cryptographic keys and passwords for each project that it manages, such as the backbone key, device authentication codes, FDSKs, management and user passwords. The UI suggests that the ETS5 would protect them with the project password, as shown in figure 29 in the appendix. The description implies that the cryptographic secrets would be encrypted with a key derived from the project password. Whenever the user tries to open the project, the ETS5 requires them to enter the project password in order to unlock it. Furthermore, a support article about the project passwords stated that “There is no way to recover your project data if the password is lost, not even via KNX (there are no hidden master keys that would make this possible).” [143]. While the article only explicitly mentioned encryption for the export, the cited statement suggests that this is generally the case, including for the locally stored projects.

However, the ETS5 does not store the cryptographic secrets securely. This was noticed when investigating the XML files that contain the project information, located in `C:\ProgramData\KNX\ETS5\ProjectStore`. The XML files are not encrypted as a whole, instead the attributes for cryptographic secrets contain Base64 encoded values, as specified in RFC 4648 [144]. When two projects use different project passwords, but for instance the same FDSK or user password, the XML files contain identical values for the respective attributes. Therefore, the project password is evidently not used to encrypt the values of passwords and keys. The project files also contain an attribute for the project password itself. Through decompiling and deobfuscating the ETS5 with ILSpy [145] and de4dot [146] it was found that a hard-coded password and salt are used to obfuscate the attributes. The password is "`ETS5Password`" and the salt is "`Ivan Medvedev`". A key and initialization vector (IV) are derived from the password and salt with `PasswordDeriveBytes` [147] from the .NET Framework. The key derivation uses SHA-1 as hash function, 100 iterations, "`ETS5Password`" as password and the ASCII byte representation of "`Ivan Medvedev`" as salt. The resulting values are shown in table 8, where the first 32 byte are used as a key and the following 16 byte as IV.

Key	`22BD16CDBB96B0E18E977BB3FEFADD8886E7E38A2F8A6FD9D2F2F5663AC20371`
IV	`8E977BB3FEFADD88E6AE6CBEAE3E7CAF`

Table 8: Key and IV for (de)obfuscation of cryptographic secrets in ETS5

The content of the attributes can be deobfuscated through the following steps:

1. The Base64 encoded attribute value is decoded according to RFC 4648 [144].
2. AES-256 with CBC cipher mode is used to decrypt the decoded value with the key and IV from table 8.
3. The PKCS #7 padding [148] is removed to receive the original value of the attribute.

The design is security by obscurity and not following best practice, as highlighted by CWE-798 and CWE-321 [135], [149]. Since the `C:\ProgramData` directory does not require elevated permissions [150] and the key and IV are static, any user or process could read the files from the project store and extract the cryptographic secrets. This is a risk, since the entire security of KNXnet/IP Secure and Data Secure hinges on the secrecy of the passwords and key material. For instance, an employee with an unprivileged user account could access the cryptographic secrets, even when the ETS5 is only installed for the administrator. Furthermore, an adversary that is able to execute code on the machine can achieve the same without requiring a privilege escalation. Knowledge of the cryptographic secrets allows taking control over the managed KNX installations.

Due to the potential security impact, the KNX Association was contacted for a coordinated vulnerability disclosure (CVD). After not receiving any reply to the support ticket for over a week, they were contacted again. In the following communication, the KNX Association held the position that it would not be the responsibility of the ETS to encrypt the keys and passwords when they are stored locally and suggested the use of BitLocker or TrueCrypt [151]. The KNX Association forwent the 90 days disclosure delay for the CVD and permitted an immediate public release of the vulnerability [151]. It is registered as CVE-2021-36799. A proof of concept is available on GitHub[2].

In follow-up emails it was explained to the KNX Association why the use of obfuscation, and misleading description of security features in the UI and support articles are harmful. Using a hard-coded key is against well-established security best practices. Customers are unaware that their cryptographic secrets are unprotected in the project store and would therefore not be able to take steps that mitigate the risk. Furthermore, full-disk encryption, as suggested by the KNX Association, would only protect the data at rest. Thus, it would not provide the same level of security as a properly encrypted storage by the ETS. After nearly four months since the initial report, the KNX Association reconsidered their position and stated that future versions of the ETS6 would encrypt the key and passwords [152]. As of ETS6 v6.0.0 the issue still exists.

Password Length	Median Hash Rate	Approx. Required Time
1	195.60 kH/s	0.0005 s
2	15276.85 kH/s	0.0005 s
3	134.25 MH/s	0.0060 s
4	290.65 MH/s	0.2800 s
5	2553.05 MH/s	3.0300 s
6	4110.50 MH/s	2 min 59 s
7	4122.50 MH/s	4 h 42 min
8	4142.75 MH/s	18 days 18 h
9	3819.40 MH/s	5 years 85 days
10*	3819.40 MH/s	496 years 277 days

Table 9: Brute-force attack against exported project
Measurements are conducted on a NVIDIA GeForce GTX 1080 Ti with hashcat 6.2.4
The measurements are rounded
*Could not be executed, extrapolation based on previous hash rate

The ETS5 also provides the option to export projects, which are supposed to be encrypted according to the support article [143]. It creates a file with the `.knxproj`

[2]`https://github.com/robertguetzkow/ets5-password-recovery`

extension, which is a ZIP-file. Within it is another ZIP-file that contains the cryptographic secrets, which is encrypted using the PKZIP stream cipher[3] and the project password as passphrase. Biham and Kocher discovered that the PKZIP stream cipher is susceptible to a known plaintext attack [73]. This attack cannot be applied, since the ETS5 compresses the files first and it only contains those with the cryptographic secrets, thus the data prior to encryption is not known. Despite the improvements to the attack by Stay [74], a known plaintext attack does not appear to be possible in this case. However, the stream cipher can be efficiently computed, thus hashcat is used to estimate the required time for brute-force and dictionary attacks. The result for brute-force attacks are shown in table 9. A single NVIDIA GeForce GTX 1080 Ti is expected to perform an exhaustive search on all 8 characters passwords within less than 19 days. An attack based on a list of 10 million common passwords would require less than a second, at a hash rate of 3819.40 MH/s. Therefore, the ETS should switch to a cryptographically secure cipher and ensure that it is less susceptible to brute-force attacks. The ETS6 supposedly implements these improvements [152].

5.4. Updating the ETS5

The ETS5 checks for updates, by downloading a manifest file from `https://update.knx.org/software/ETS50/MainManifest.xml` through TLS v1.2. Within the file is a RSASSA-PKCS #1 v1.5 signature, as specified in RFC 8017 [153], with which the ETS5 verifies the authenticity of the content. Contained in the embedded resources of the `Knx.Updater.dll` is the public key that is required to check the signature. The `MainManifest.xml` points to the URL of another manifest file for the most recent version of the ETS5, which in turn contains the URL of the executable. Included in the second manifest XML are both a signature for its content and the executable. When the ETS5 performs an update, it verifies the signature of the manifest, downloads the executable, verifies its signature and runs the executable.

There does not appear to be a practical way to attack the update process, assuming that the web server is secure and the implementation of TLS v1.2 used by the ETS5 is not vulnerable against an attack that permits to impersonate the server. However, future versions of the ETS should not be using SHA-1 for the signatures, given the improvements in collision attacks against it by Leurent and Peyrin [154] [155].

5.5. Analysis with Test Software

A test software was developed to evaluate the conformance of KNXnet/IP Secure clients and servers with the ISO 22510:2019 standard. Test cases are constructed through finite state machines, which determine when and what kind of frames are sent by the test software. It permits constructing arbitrarily modified KNXnet/IP Secure frames. This allows to build tests that check requirements set forth by the standard, evaluate the behavior for edge cases and boundary values that are ambigu-

[3]It is also known as ZipCrypto or PKWARE encryption

ously specified, and deliberately construct incorrect frames to analyze the reaction of the test target. A full list of the implemented tests can be found in section "A.4 Test Cases for the ETS5 and KNXnet/IP Secure Routers" of the appendix. In this section the test software is applied to the ETS5, which has to implement the client-side of the KNXnet/IP Secure protocol as specified in ISO 22510:2019.

In the test setup the ETS5 software is running in a virtual machine and the test software on the host of the virtual machine. The network configuration permits the communication between both of them. A tunneling connection can be established through the UI of the ETS5 in the "Bus" tab. For each test case the respective implementation by the test software is started, listening on the well-known port 3671 for incoming requests, and then the creation of a new connection by the ETS5 with the test software is manually triggered. Frames sent and received are documented in a log file.

The results of eleven tests conducted with the ETS5 are shown in table 10. A smaller set of tests was applied to the ETS5, compared to the certified devices in "6.1 Analysis with Test Software", because the ISO 22510:2019 standard mainly specifies requirements for the server, while leaving correct behavior for the client open to interpretations in several areas. Ten tests were successfully completed by rejecting invalid frames, matching the specification in ISO 22510:2019. Details about these test results can be found in table 18 of the appendix. However, there are problems that have been discovered while implementing the test software and one of the tests failed.

Target	**Test No.**										
	1	2	3	4	5	6	7	8	9	10	11
ETS5	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 10: Test results for the ETS5
Symbols: ✓fulfilled, × unfulfilled; – not applicable
See table 17 and 18 for test details.

An unexpected issue is that the ETS5 attempts to establish the KNXnet/IP Secure unicast connection through UDP instead of TCP. ISO 22510:2019 expressly forbids the use of UDP for KNXnet/IP Secure unicast [6, p. 122, p. 126]. The implementation of the test software had to be modified, so that it processes KNXnet/IP Secure unicast frames received through UDP. Additionally, the ETS5 does not retrieve the required key material for the authentication from the available projects, when establishing connections in the "Bus" tab. Instead, the user has to manually enter the password for the tunneling user and optionally the password from which the device authentication code is derived. ISO 22510:2019 permits the client to skip the authentication of the server [6, p. 127]. The ETS5 makes use of this provision, which has been previously criticized in section "4.1.2 Authenticated Key Exchange and Session", due to its detrimental effect on the security of the protocol.

In the first test, the ETS5 and test software conduct the handshake for establishing a KNXnet/IP Secure session. The ETS5 sends a `SESSION_REQUEST`, which the test software replies to with a `SESSION_RESPONSE`. The ETS5 then attempts to authenticate itself with a `SECURE_WRAPPER` containing the `SESSION_AUTHENTICATE` frame. In the test the response to this frame is an unencapsulated `SESSION_STATUS` frame with the status code `STATUS_AUTHENTICATION_SUCCESS`, which is incorrect and should be rejected by the client. However, the ETS5 still proceeds with a `SECURE_WRAPPER` containing a `CONNECT_REQUEST`. This means that either the ETS5 has processed the reply, considering it valid, or it did not wait for a confirmation of the successful authentication. The behavior is incorrect in both cases.

In summary, the ETS5 appears to mostly follow the standard in the conducted tests, but the use of UDP and the processing of the authentication confirmation differ significantly from the specification. However, none of the violations of ISO 22510:2019 seem to have an impact on the security. The logs for all tests are included in the digital files, see "A.7 Software, Models and Logs" in the appendix.

5.6. Improvement Suggestions

Due to the delayed handling of the initial report for the CVD, it would be recommended that the KNX Association establishes a dedicated contact for security issues to ensure a timely response and triaging of the report by domain experts. Furthermore, security engineering should implement a thorough review process for the application design of the ETS, in order to verify that the concepts do not rely on security by obscurity and best practices for application security are being followed. The use of a hard-coded password and salt to obfuscate security critical cryptographic secrets is not an acceptable solution, as it attempts to create the impression that data would be protected, while it does not provide any actual security benefits over storing the values as plaintext. Additionally, the documentation and UI should transparently communicate the security features or lack thereof to the user. This ensures that they can make an informed decision. The responsibility for handling security shortcomings in the ETS should not be shifted onto the users, like the KNX Association initially did, especially when they are not being informed about them and thus unable to set up mitigations. The responsibility lies with the KNX Association to address the underlying issue through proper software engineering, as they have agreed to for future versions of the ETS, after the CVD. If the KNX Association remains receptive to feedback provided by security researchers, this will be beneficial to the quality and security of their products.

The storage of cryptographic secrets could be improved by encrypting the XML files with a cipher mode that provides AEAD and deriving the key from the project password with a modern key derivation function, such as Argon2id [131]. Similarly, the exported projects should move away from the insecure PKZIP stream cipher, which is already supposed to be the case in the ETS6, but was not evaluated as part of this thesis. The security of KNXnet/IP Secure, as used in practice, could be significantly improved by increasing the default length of the generated passwords and ensuring that the ETS always authenticates the server through the device authentication code. This information about the server has to be available, since establishing the connection also requires one of the user passwords. Ideally, the ETS would automatically retrieve the needed key material from the project files, without requiring manual input by the user. Throughout the ETS all uses of SHA-1 for security relevant applications should be replaced with a cryptographic hash function that is currently resistant to chosen-prefix attacks.

6. Analysis of Certified Devices

The KNX Association requires devices to pass a certification process, before manufacturers are permitted to use the KNX trademark on their product [156]. While the exact tests for KNXnet/IP Secure are not publicly documented, certified devices should conform with ISO 22510:2019. In this chapter two certified KNXnet/IP Secure routers, MDT SCN-IP100.03 and Weinzierl KNX IP Router 752 Secure, are tested for their conformance with the standard. The purpose is to identify potential deviations from ISO 22510:2019, that may have an impact on the security of the product or indicate missing coverage in the testing of the certification process. The conformance is evaluated through black-box tests, conducted with the software previously introduced in “5.5 Analysis with Test Software”, protocol state fuzzing to determine a FSM that models the observable behavior of each device, and model checking with NuXMV to compare the inferred state machines with the session FSM from ISO 22510:2019.

6.1. Analysis with Test Software

The analysis with the test software was conducted after the certified KNXnet/IP Secure routers have been commissioned with the ETS5, following the instructions provided by the manufacturers. The devices were connected to the local network, listening for incoming requests on port 3671. Routers have to implement the server-side of the KNXnet/IP Secure protocol specified in the ISO 22510:2019 standard. The test software was running on a computer in the local network, acting as a client by initiating the communication with the target devices. The results of the 27 tests conducted with the KNXnet/IP Secure routers are shown in table 11 and 12. The logs for all tests are included in the digital files, see “A.7 Software, Models and Logs” in the appendix. Comparing the overall test results, MDT SCN-IP100.03 passed 22 tests successfully, while the Weinzierl KNX IP Router 752 Secure fulfilled 18 tests. Details about the successful tests are provided in table 19 in the appendix. The ambiguous and failed tests require an evaluation of the individual cases to assess their potential impact on the security and the conformance with ISO 22510:2019. Results that are considered ambiguous may not be the fault of the manufacturer, but rather caused by an imprecise specification that permits different equally valid interpretations.

In test 1, a KNXnet/IP Secure session is established, but prior to the client authentication a `SECURE_WRAPPER` is sent, containing a `CONNECT_REQUEST` for a KNXnet/IP connection. The KNX IP Router 752 Secure incorrectly responds with a `SESSION_-STATUS` containing the status code `STATUS_TIMEOUT`. According the session FSM in figure 19, the correct reply in state UNAUTHENTICATED would be a SESSION_STATUS with status code STATUS_UNAUTHENTICATED, as shown by event E05 and action A6.

The reaction to frames with invalid headers is checked in tests 7, 8, 9 and 10. The ISO 22510:2019 states, that “If the header is not a well-formed KNXnet/IP header, the receiver shall close the TCP connection” [6, p. 42]. In the cases marked as ambiguous in table 11, the connection was eventually closed after a timeout, but not immediately. However, the standard does not specify that the TCP connection has to be closed immediately nor whether well-formed refers to the structure of the header or also the values of the fields. The structure of the frames sent by the test software matches the format of the KNXnet/IP header, but the length, service type and protocol version have been altered to invalid values in the respective tests.

SCN-IP100.03 fails test 10 with firmware v3.0.3. It becomes unresponsive to KNXnet/IP Secure frames after receiving a SESSION_REQUEST with the length field set to a value larger or equal to 0x0259. A reboot of the devices is required to restore normal operation. This is a DoS vulnerability in firmware v3.0.3, as it allows an adversary to effectively disrupt the operation of the device. The bug was reported to the manufacturer MDT in a CVD, which was handled very professionally. MDT confirmed the bug and provided a patched firmware for testing, to confirm that the issue was solved. An official release of the updated firmware was published shortly afterwards. The vulnerability is registered as CVE-2021-37740, which was fixed by firmware v3.0.4.

In test 11, it is evaluated whether the devices reject frames with previous sequence numbers. This works properly with the KNX IP Router 752 Secure, except when the sequence number is never incremented and remains at zero. Subsequent SECURE_-WRAPPER frames that also contain the sequence number of zero are accepted. This is a violation of the specification, which demands that “[…] frames shall be discarded if the sequence number is less than or equal to the last received number” [6, p. 108].

Test 12 checks the reaction to an encapsulated SESSION_STATUS frame with an invalid reserved field. Both devices accept the frame, but this is technically correct, because the ISO 22510:2019 fails to specify requirements for the validation of SESSION_STATUS frames.

In test 13, a KNXnet/IP Secure session is established and for the authentication the SESSION_AUTHENTICATE frame is modified to contain an invalid value in its reserved field. Both devices reject the frame, but they respond with a SECURE_WRAPPER containing a SESSION_STATUS with the status field set to STATUS_TIMEOUT. This is incorrect, since a valid SECURE_WRAPPER containing an invalid SESSION_AUTHENTICATE frame should trigger event E02 and action A2, when in state UNAUTHENTICATED [6, pp. 123-124]. The correct reply would be a SECURE_WRAPPER containing a SESSION_STATUS with the status field set to STATUS_AUTHENTICATION_FAILED.

Test 24 and 25 check the access control. Both devices correctly reject attempts to create device management connections through either plain KNXnet/IP or with a non-management users in KNXnet/IP Secure sessions. However, both devices uses different status codes in their CONNECT_RESPONSE and ISO 22510:2019 does not precisely define which is the correct one. The SCN-IP100.03 rejects the connection attempts with an E_AUTHORISATION_ERROR. According to ISO 22510:2019, the status code is meant to be sent when a client is not authorized to use the IA requested in the extended connection request information (CRI) [6, p. 152]. However, no extended CRI was used. The KNX IP Router 752 Secure rejects the connection attempts with a `E_NO_MORE_CONNECTIONS` status code. According to ISO 22510:2019, the status code is meant to be sent when the server is unable to accept new connections because the maximum number of concurrent connections is already active [6, p. 152]. However, there is no concurrent connection and one management connection would be supported. It appears that the description of the status codes in ISO 22510:2019 is inaccurate or incomplete. Presumably the `E_AUTHORISATION_ERROR` would be the correct status code for this test case.

An additional test was conducted to evaluate how the devices handle the multicast timer reaching its maximum value. When the SCN-IP100.03 receives a valid `TIMER_NOTIFY` frame with the maximum timer value, the timer does not wrap-around, it remains stuck at the maximum. The KNX IP Router 752 Secure performs a wrap-around. As previously explained in "4.2.1 CCM Requirements", both are valid implementations, as the standard does not specify how to handle the case, but the wrap-around would be preferable to avoid attacks against the confidentiality due to nonce reuse.

In summary, the KNXnet/IP Secure routers mostly conform with the ISO 22510:2019 standard in the tested cases. Important functionality such as access control and authorization could not be bypassed. Even though a DoS vulnerability was found in the SCN-IP100.03, it was quickly patched and the manufacturer demonstrated that they take their product's security serious. Deviations from the standard exist, but they do not appear to affect the security.

Device	**Test No.**													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SCN-IP100.03	✓	✓	✓	✓	✓	✓	✓	○	✓	×	✓	✓	×	✓
KNX IP Router 752 Secure	×	✓	✓	✓	✓	✓	○	○	○	○	×	✓	×	✓

Table 11: Test results for the KNXnet/IP Secure routers - Part 1
Symbols: ✓fulfilled, × unfulfilled; ○ ambiguous
See table 17 and 19 for test details.

Device	**Test No.**												
	15	16	17	18	19	20	21	22	23	24	25	26	27
SCN-IP100.03	✓	✓	✓	✓	✓	✓	✓	✓	✓	○	○	✓	✓
KNX IP Router 752 Secure	✓	✓	✓	✓	✓	✓	✓	✓	✓	○	○	✓	✓

Table 12: Test results for the KNXnet/IP Secure routers - Part 2
Symbols: ✓fulfilled, × unfulfilled; ○ ambiguous
See table 17 and 19 for test details.

6.2. Protocol State Fuzzing and Model Checking

While the test software from the previous section permits constructing specialized tests, where the content of frames can be controlled down to the bit, they all have to be handcrafted. Thus, the approach provides highly specific insights, but a limited coverage of the target device's overall behavior. Protocol state fuzzing is an automated approach, that can be applied to each KNXnet/IP Secure router to infer a FSM, that models its observable behavior. This allows a comparison with the session FSM specified in ISO 22510:2019. The general process and implementation was previously explained in "3.5 Protocol State Fuzzing". For this work the state learner by de Ruiter and Poll [81], [82] has been adapted to support KNXnet/IP Secure. It implements the optimization idea of de Ruiter and Poll for the W-method, by stopping the search for a counterexample in a trace when the KNXnet/IP Secure session or TCP connection is closed by either party. An additional constraint is integrated, to ensure that only one session is created at a time. A TCP connection could contain multiple concurrent KNXnet/IP Secure sessions, but the specified session FSM is for one session only. The alphabet includes both representations of valid and invalid frames. Thus, the reactions to incorrect frames by the KNXnet/IP Secure routers are tested as well. The resulting FSMs from the protocol state fuzzing were translated into the NuXMV syntax for model checking. The NuXMV model of the session FSM from "4.1.4 Model Checking of Session FSM" was build with the same alphabet to facilitate a direct comparison. The source code of the state learner and NuXMV models are provided in the digital files, see "A.7 Software, Models and Logs" in the appendix.

The FSMs that were determined by the state learner are depicted in figures 27 and 28. Evaluating which parts fulfill the ISO 22510:2019 standard was accomplished through model checking and manual comparison with the standard. Parts marked in purple are correctly implemented by the KNXnet/IP Secure routers, but deviate from the notation of the session FSM in figure 19, because protocol state fuzzing is unable to infer the existence and use of internal state variable, such as timers. Therefore, it models the states explicitly, when a changed behavior of the SUT is observed. The purple transitions also include cases that are not depicted in the session FSM of ISO 22510:2019, such as the termination of the connection when the KNXnet/IP header is not well-formed.

It can be seen that both the MDT SCN-IP100.03 and Weinzierl KNX IP Router 752 Secure mainly deviate from the standard during error handling for invalid frames. None of the identified issues permit to bypass the authentication. They only do not immediately terminate the session under all circumstance, when they are supposed to. The device by MDT follows ISO 22510:2019 more closely. Its main difference to the session FSM is, that frames with an incorrect HPAI appear to result in a state where the TCP connection still exists, but the communication partner is unable to establish as session. A return to the idle state 0 happens when the TCP connection is terminated, either through a timeout or when a frame with an ill-formed KNXnet/IP header is sent. This is why SCN-IP100.03 fails test 2, as shown in table 13, because frames other than `SESSION_REQUEST` result in a transition away from state 0. The

KNX IP Router 752 Secure fails both tests 2 and 14, because it does not immediately terminate the TCP connection when receiving a frame with an invalid header. Instead, it transitions to an intermediate state 6 until a timeout occurs or the connection is explicitly closed. As explained in the previous section, both devices do not respond with an encapsulated `SESSION_STATUS` with the status field set to `STATUS_-AUTHENTICATION_FAILED`, when they received an `SESSION_AUTHENTICATE` frame with an invalid reserved field. Thus, both devices fail test 9. However, thereby they fulfill the textual description from ISO 22510:2019 and test 19, which are contradicting the definition of the session FSM, as previously discussed in “4.1.4 Model Checking of Session FSM”.

The KNX IP Router 752 Secure appears to deviate from the standard when handling invalid and unexpected frame, more so than the SCN-IP100.03. When comparing the FSMs in figure 27 and 28, the issues are more significant for KNX IP Router 752 Secure, because it does not immediately transition to state 0 in cases that are well-defined in both the text and session FSM in ISO 22510:2019. For instance, when the FSM is in state 1 (`UNAUTHENTICATED`) and the device receives an encapsulated `SESSION_AUTHENTICATE` with an invalid MAC, then it should deallocate the session and transition into state 0 (`IDLE`) [6, pp. 123-124]. This is not the case and similar issues exist with requests in `SECURE_WRAPPER` frames prior to authentication, encapsulated `SESSION_STATUS` frame with the status code `KEEP_ALIVE`, and incorrect user IDs in `SESSION_AUTHENTICATE` frames. This is why test 1 of the model checking fails for this device.

Model	**Test No.**											
	1	2	3	4	5	6	7	8	9	10	11	12
Session FSM	✓	✓	✓	–	–	×	✓	✓	✓	✓	✓	✓
SCN-IP100.03	✓	×	✓	–	–	×	✓	–	×	✓	✓	✓
KNX IP Router 752 Secure	×	×	✓	–	–	×	✓	–	×	✓	✓	✓

Table 13: Model checking inferred session FSMs - Part 1
Symbols: ✓fulfilled, × unfulfilled; – not applicable
See table 15 for test details.

Model	**Test No.**										
	13	14	15	16	17	18	19	20	21	22	23
Session FSM	–	○	✓	✓	✓	✓	×	✓	✓	×	✓
SCN-IP100.03	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
KNX IP Router 752 Secure	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 14: Model checking inferred session FSMs - Part 2
Symbols: ✓fulfilled, × unfulfilled; – not applicable
See table 15 for test details.

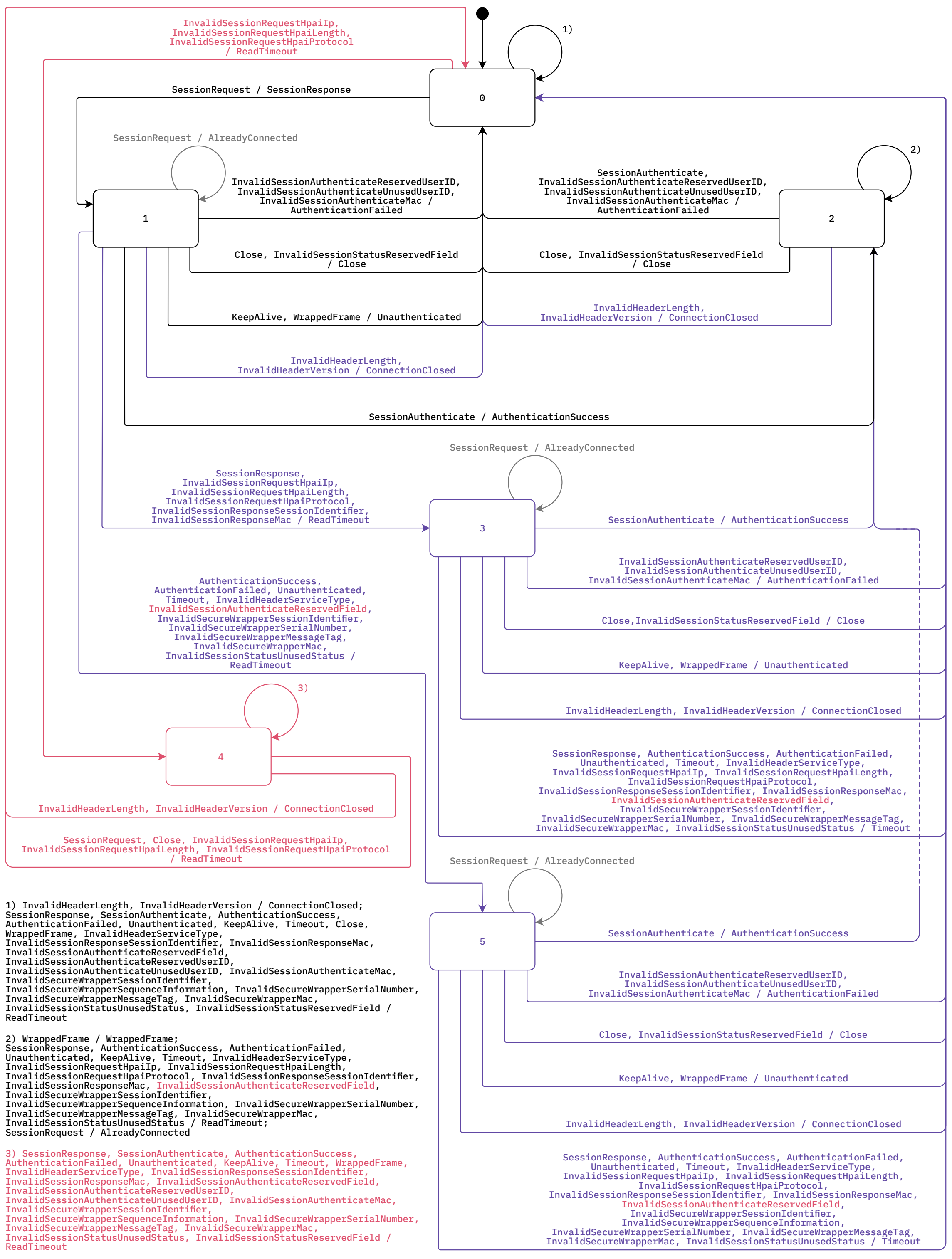

InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol / ReadTimeout
1)
0
SessionRequest / SessionResponse
SessionRequest / AlreadyConnected
2)
InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed
SessionAuthenticate, InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed
1
2
Close, InvalidSessionStatusReservedField / Close
Close, InvalidSessionStatusReservedField / Close
KeepAlive, WrappedFrame / Unauthenticated
InvalidHeaderLength, InvalidHeaderVersion / ConnectionClosed
InvalidHeaderLength, InvalidHeaderVersion / ConnectionClosed
SessionAuthenticate / AuthenticationSuccess
SessionRequest / AlreadyConnected
SessionResponse, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac / ReadTimeout
3
SessionAuthenticate / AuthenticationSuccess
InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed
AuthenticationSuccess, AuthenticationFailed, Unauthenticated, Timeout, InvalidHeaderServiceType, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / ReadTimeout
Close,InvalidSessionStatusReservedField / Close
KeepAlive, WrappedFrame / Unauthenticated
3)
InvalidHeaderLength, InvalidHeaderVersion / ConnectionClosed
4
SessionResponse, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, Timeout, InvalidHeaderServiceType, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / Timeout
InvalidHeaderLength, InvalidHeaderVersion / ConnectionClosed
SessionRequest, Close, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol / ReadTimeout
SessionRequest / AlreadyConnected
5
SessionAuthenticate / AuthenticationSuccess
InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed
Close, InvalidSessionStatusReservedField / Close
KeepAlive, WrappedFrame / Unauthenticated
InvalidHeaderLength, InvalidHeaderVersion / ConnectionClosed
SessionResponse, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, Timeout, InvalidHeaderServiceType, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSequenceInformation, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / Timeout
1) InvalidHeaderLength, InvalidHeaderVersion / ConnectionClosed; SessionResponse, SessionAuthenticate, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, KeepAlive, Timeout, Close, WrappedFrame, InvalidHeaderServiceType, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSequenceInformation, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus, InvalidSessionStatusReservedField / ReadTimeout
2) WrappedFrame / WrappedFrame; SessionResponse, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, KeepAlive, Timeout, InvalidHeaderServiceType, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSequenceInformation, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / ReadTimeout; SessionRequest / AlreadyConnected
3) SessionResponse, SessionAuthenticate, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, KeepAlive, Timeout, WrappedFrame, InvalidHeaderServiceType, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSequenceInformation, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus, InvalidSessionStatusReservedField / ReadTimeout

Figure 27: Result of protocol state fuzzing for SCN-IP100.03
Black - Correct, matches session FSM
Purple - Correct, but notation deviates from session FSM
Red - Incorrect, violates specification

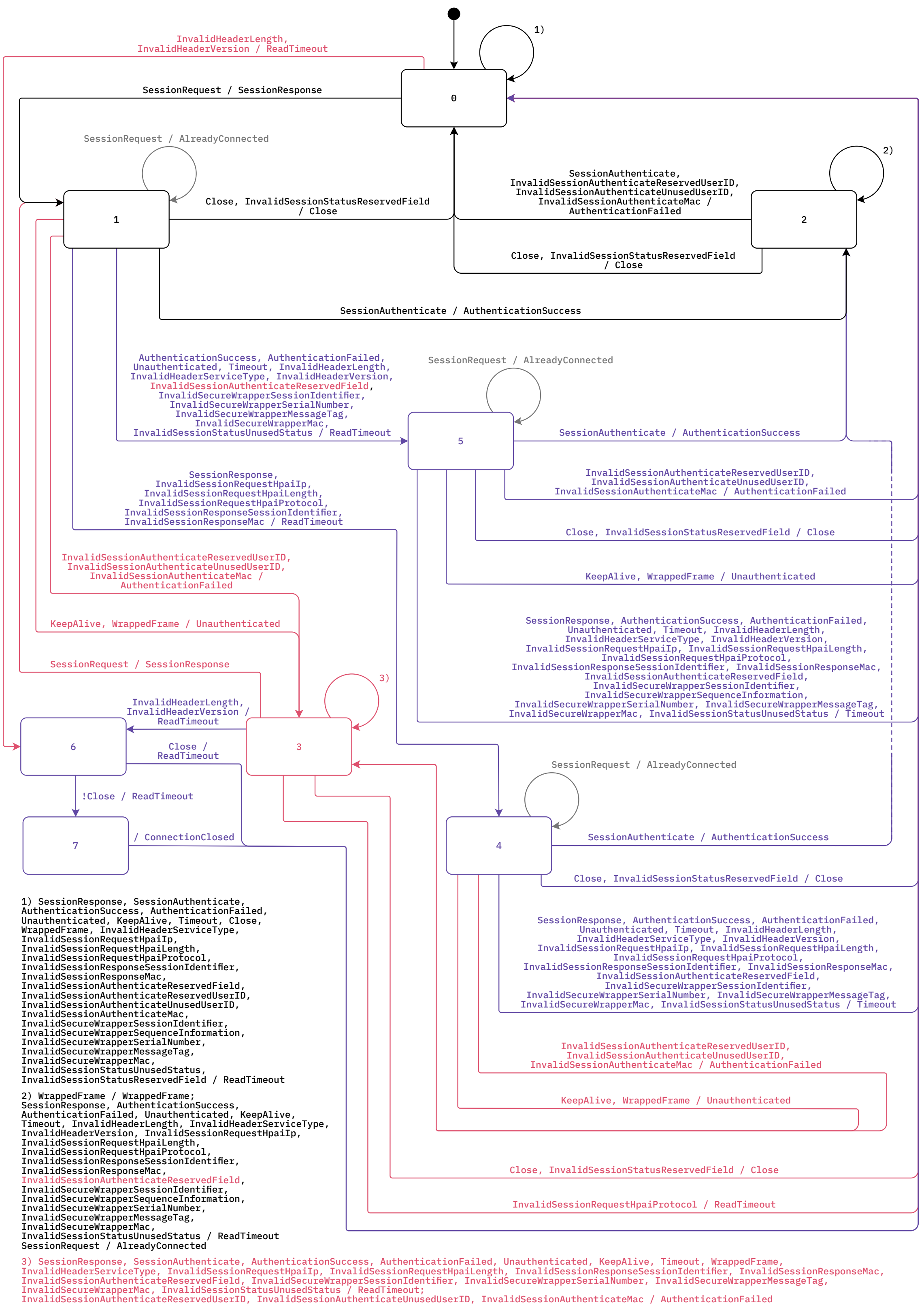

1)
InvalidHeaderLength, InvalidHeaderVersion / ReadTimeout
SessionRequest / SessionResponse
0
SessionRequest / AlreadyConnected
Close, InvalidSessionStatusReservedField / Close
SessionAuthenticate, InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed
2)
1
2
Close, InvalidSessionStatusReservedField / Close
SessionAuthenticate / AuthenticationSuccess
AuthenticationSuccess, AuthenticationFailed, Unauthenticated, Timeout, InvalidHeaderLength, InvalidHeaderServiceType, InvalidHeaderVersion, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / ReadTimeout
SessionRequest / AlreadyConnected
5
SessionAuthenticate / AuthenticationSuccess
InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed
SessionResponse, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac / ReadTimeout
Close, InvalidSessionStatusReservedField / Close
InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed
KeepAlive, WrappedFrame / Unauthenticated
KeepAlive, WrappedFrame / Unauthenticated
SessionResponse, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, Timeout, InvalidHeaderLength, InvalidHeaderServiceType, InvalidHeaderVersion, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSequenceInformation, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / Timeout
SessionRequest / SessionResponse
3)
InvalidHeaderLength, InvalidHeaderVersion / ReadTimeout
6
3
Close / ReadTimeout
SessionRequest / AlreadyConnected
!Close / ReadTimeout
7
/ ConnectionClosed
4
SessionAuthenticate / AuthenticationSuccess
Close, InvalidSessionStatusReservedField / Close
SessionResponse, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, Timeout, InvalidHeaderLength, InvalidHeaderServiceType, InvalidHeaderVersion, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / Timeout
InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed
KeepAlive, WrappedFrame / Unauthenticated
Close, InvalidSessionStatusReservedField / Close
InvalidSessionRequestHpaiProtocol / ReadTimeout
1) SessionResponse, SessionAuthenticate, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, KeepAlive, Timeout, Close, WrappedFrame, InvalidHeaderServiceType, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSequenceInformation, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus, InvalidSessionStatusReservedField / ReadTimeout
2) WrappedFrame / WrappedFrame; SessionResponse, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, KeepAlive, Timeout, InvalidHeaderLength, InvalidHeaderServiceType, InvalidHeaderVersion, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionRequestHpaiProtocol, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSequenceInformation, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / ReadTimeout SessionRequest / AlreadyConnected
3) SessionResponse, SessionAuthenticate, AuthenticationSuccess, AuthenticationFailed, Unauthenticated, KeepAlive, Timeout, WrappedFrame, InvalidHeaderServiceType, InvalidSessionRequestHpaiIp, InvalidSessionRequestHpaiLength, InvalidSessionResponseSessionIdentifier, InvalidSessionResponseMac, InvalidSessionAuthenticateReservedField, InvalidSecureWrapperSessionIdentifier, InvalidSecureWrapperSerialNumber, InvalidSecureWrapperMessageTag, InvalidSecureWrapperMac, InvalidSessionStatusUnusedStatus / ReadTimeout; InvalidSessionAuthenticateReservedUserID, InvalidSessionAuthenticateUnusedUserID, InvalidSessionAuthenticateMac / AuthenticationFailed

Figure 28: Result of protocol state fuzzing for KNX IP Router 752 Secure
Black - Correct, matches session FSM
Purple - Correct, but notation deviates from session FSM
Red - Incorrect, violates specification

6.3. Improvement Suggestions

It would be recommended that manufacturers apply white-box or gray-box fuzzing techniques in order to identify bugs, such as the one that caused CVE-2021-37740, during development. The results of the test software, protocol state fuzzing and model checking have shown, that the certification process does not ensure complete conformance with the ISO 22510:2019 standard. Therefore, the requirements and test suite should be extended to cover cases that the certified devices did not fulfill. Protocol state fuzzing could also be applied as part of the process. Furthermore, the ambiguities and inconsistencies of the protocol specification should be addressed, as it creates unnecessary challenges for the implementation and testing.

7. Risk Analysis with BSI 200-3

In order to systematically identify risks that KNX installation could be exposed to, the BSI 200-3 standard was applied to three fictional KNX installations that are based on real showcases from the KNX journals. The risk analysis was conducted on fictional KNX installations, because the KNX journals and system integrators do not provide in-depth details, such as network plans, about real installations. Additionally, the examples could be kept minimal and it was possible to intentionally integrate risk factors to consider how they affect the KNX installation and connected network. The three KNX installations portrait different use cases, which are in a house, a hotel and an art gallery. Each of them has different security requirements and risks that they are exposed to. Inspiration for the house is provided by the "Beverly Hills Mansion" showcase [23, pp. 14-15] and the hotel is based on the "Dubai Parks and Resorts" [23, pp. 12-13], both are from the second KNX journal of 2018. The art gallery is based on the "Institute of Greek Artists and Scientists of the Diaspora" from KNX journal of 2020 [11, pp. 24-25]. BSI 200-3 and the IT-Grundschutz compendium were applied for the risk analysis, as previously described in "3.6 Risk Analysis with BSI 200-3". Due to length limits, this thesis presents only the highly summarized results of the risk analysis, which identify common risks that are applicable to nearly all KNX installations that use KNXnet/IP Secure. Details can be found in the digital files, see "A.7 Software, Models and Logs" in the appendix. Based on the results, it can be evaluated whether the guidelines provided by the KNX Association in the "KNX Project Preparation" document [157] adequately address the identified risks or whether amendments would be recommended.

7.1. Results of the Risk Analysis

While KNXnet/IP Secure addresses shortcomings of KNXnet/IP and thereby improves the security of KNX installations, it does not make them fully secure. Even in the best-case scenario, it only protects the communication between KNXnet/IP Secure devices, ensures access control for their services, and limits configuration to an authorized management user. There may still be a significant remaining attack surface against the devices, ETS, and network that needs to be addressed as well. This is both relevant for protecting the KNX installation from risks from the network it is connected to and vice versa. The most relevant risks for the secure operation of a KNX installation and the surrounding network, that have been identified through BSI 200-3, are explained in the following paragraphs.

An insufficient security concept for the network architecture could expose the KNX installation to attacks from the internet or threats within the local network. For instance, missing segmentation, misconfigured firewalls or a lack of network access control (NAC) could allow unauthorized devices in the network to communicate with security critical ones. This is especially relevant for companies, when the KNX installation interfaces with surveillance cameras, alarm systems and electronic door locks. In the worst-case KNX devices would be directly reachable from the internet. On March 20, 2022, that is the case for 15534 KNXnet/IP devices globally, according to the Shodan search engine[4]. Furthermore, a lack of monitoring in the network poses a risk, because security incidents might not be noticed.

Unpatched vulnerabilities in operating systems, software and firmware may allow an adversary to attack KNXnet/IP devices or the computer running the ETS software. As shown in CVE-2021-37740, a DoS vulnerability could be used to disrupt the operation of the KNX installation. Severe vulnerabilities, such as those permitting remote code execution (RCE), could be an attack vector for malware. Embedded and IoT devices have been targeted in the past, for instance by the Mirai botnet, as explained by Antonakakis et al. [158]. It is therefore conceivable, that KNXnet/IP devices, that are connected to the internet, could become a target of similar malware campaigns in the future. Since the ETS does not properly encrypt the key material, an infection of the computer would allow an adversary to exfiltrate the secrets, thus gaining full control over the KNX installation. A different risk is data loss due to ransomware or hardware failure. It could cause an irretrievable loss of project files, which would prevent the management of KNX installations. Therefore, another risk is the lack of backups, which would be necessary to recover the data. Without backups, all KNX devices would have to be reset and the entire ETS project recreated, should modifications to the KNX installation be required.

As previously discussed in "5.2 Offline Attack against Authentication", selecting weak passwords for the AKE in KNXnet/IP Secure may allow an adversary to determine the password and impersonate a legitimate device. In case they are able to impersonate a client, they may be able to control actuators in the KNX installation, similar to Molina et al. [18].

Placing KNXnet/IP devices in publicly accessible areas, thereby lacking physical security, is mainly a concern for companies. This threat is particularly relevant for hotels, where KNX devices might be placed inside hotel rooms, where an attacker would have undisturbed access to them, without facing camera surveillance.

Companies such as Gira and Jung sell cloud services for remote management [159], [160]. Considering that the planned lifetime of a building automation system in pri-

[4]`https://www.shodan.io/search?query=KNXnet%2FIP`

vate homes could be upwards of a decade, it could be counterproductive to create dependencies to potentially short-lived cloud services. This may affect the availability of important functions that the customer grows to rely on. Furthermore, remote access is highly sensitive for the security of the computer network. Vulnerabilities affecting the cloud service or gateway can have a significant security impact on their customers. Depending on the implementation, it may also raise privacy concerns.

Lastly, the system integrator that installs and configures the KNX installation has access to the project files and their key material. It could be a risk factor, that the files are stored by the company. It may become a target by adversaries that try to gain knowledge about their customer's installations and cryptographic keys. Additionally, once the project is finished, the system integrator has to hand over the project files to the customer. If this is done through insecure means, for instance by sending an unencrypted email with the exported project and perhaps even the project password, then a compromise of the email account would also affect the security of the KNX installation.

7.2. KNX Guidelines

The KNX Associations provides a checklist for system integrators to increase the security and privacy in KNX installations [157, pp. 28-30]. It is fairly comprehensive and contains mostly good advice. It thoroughly addresses physical security measurements. For KNXnet/IP it also requires the documentation and handover of network settings to the customer [157, p. 29], which is commendable. However, it also contains questionable security suggestions, which include MAC filtering and changing the default multicast IP address [157, p. 29]. While they are not harmful, they are not effective measurements for access control. Unfortunately, the guide does not provide proper suggestions for NAC. Furthermore, the checklist recommends a separate LAN for the KNX communication [157, p. 29]. Air-gapping the KNX installation would be great for security, but the recommendation is too simplistic and contrary to nearly all marketed use case that require connections to other device on the network. The checklist is missing steps to ensure that the KNX installation can be safely integrated into an existing network architecture, with appropriate security measurements.

Additional good guidelines include the use of strong passwords and preventing KNX devices from accessing the internet [157, p. 29]. This addresses two central security concerns. However, the latter may be at odds with the KNX Association's push towards IoT, as described in the KNX journal of 2021 [161]. The guide also mentions the case where KNX is interfacing with security systems and says to ensure that KNX is unable to trigger security relevant functions in the connected system [157, p. 30]. This limits the security impact should KNX devices be compromised. Furthermore, keeping the ETS, operating system and device firmware updated is also advised [157, p. 30]. This addresses the previously identified risk. It warns against connecting untrusted media, such as USB sticks [157, p. 30], which is sensible. However, it also advises to save the project file after the installation to a secured USB stick and delete the project from the computer [157, p. 30]. USB sticks may not be the most suitable media for long-term archival. Since the guide is aimed at system integrators, the deletion from the computer is a good step to ensure that only the customer has access to the key material after handing over the project files.

In summary, the checklist provides several good recommendations, but it is not complete.

7.3. Improvement Suggestions

In order to improve the security of the KNX installation and network further, the following steps are recommended:

- The computer with the ETS should solely be used for managing the KNX installation, not as a PC. It should not be used to browse the internet, read emails or install software from non-reputable sources. Unless administrative privileges are required, a regular user account should be utilized.
- Full-disk encryption should be applied to protect data at rest, such as the key material of the ETS5.
- Since the ETS5 does not provide an option to auto-generate passwords of maximum length, a password manager should be used. This is generally recommendable for securely storing and generating passwords.
- The certificate sticker should be removed from all KNXnet/IP Secure devices, as it contains the FDSK.
- Under no circumstance, not even for testing, should KNX devices be connected to the internet through port forwarding. Additionally, UPnP should be disabled, if possible.
- Backups to suitable media should be made and regularly checked for successful recovery.
- The network architecture should be designed with security in mind. Therefore, segmentation and NAC has to be included, either through a classic zone-based concept, such the one detailed in the ISi-LANA standard by the BSI [92], or a modern Zero Trust design as described in NIST SP 800-207 [94]. While not support by KNX itself, multi-factor authentication (MFA) should be used where possible.
- For companies a security information and event management (SIEM) should be implemented to monitor and evaluate events in the network to detect potential security incidents. Intrusion detection systems (IDSs) and anti-virus at endpoints may help to detect and isolate threats as well.
- Ideally, cloud services for remote management should be avoided.

8. Evaluation

The research questions posed in the “Introduction” chapter were all answered.

1. The unicast and multicast protocol, specified in ISO 22510:2019, are not secure from a theoretical standpoint. Multiple design flaws that affect the security properties were pointed out in chapter “Analysis of KNXnet/IP Secure in ISO 22510:2019”, including nonce reuse and the possibility of offline attacks against the authentication of KNXnet/IP Secure unicast.
2. The offline attack against the authentication could be a practical, as shown in “5.2 Offline Attack against Authentication”, depending on the password complexity and available computational resources. Furthermore, the compromise of the cryptographic secrets from one devices would allow to impersonate others, due the protocol relying on symmetric cryptography for the authentication.
3. The ETS5 v5.7.6 uses a hard-coded password and salt for the obfuscation of cryptographic secrets in the project store. CVE-2021-36799 puts the security of the KNX installation at risk, because the cryptographic secrets can be retrieved without knowledge of the project password, contrary to what the UI and documentation suggests.
4. The tests in “5.5 Analysis with Test Software” show that the ETS5 v5.7.6 does not fully conform with ISO 22510:2019.
5. The two tested KNXnet/IP Secure routers did not fully conform with ISO 22510:2019, as show in “Analysis of Certified Devices”.
6. Non-conforming behavior did not have a negative impact on the security, except for the DoS vulnerability CVE-2021-37740, which was reported to the manufacturer and fixed.
7. Risks for KNX installations and their mitigations were identified in “Risk Analysis with BSI 200-3”, as well as improvement suggestions for the guidelines provided by the KNX Association.

9. Conclusion and Future Work

In conclusion, the unicast and multicast KNXnet/IP Secure protocols improve the security in KNX installations compared to communication over plain KNXnet/IP, but from a theoretical standpoint they are neither secure nor well-designed cryptographic protocols. A practical attack against the authentication of the unicast protocol is possible when weak passwords are used. However, the demonstrated attack is not possible for randomly generated passwords with 10 or more characters. Ideally, the discovered design flaws in this work and those found by Judmayer et al. [1] would be used to develop a new version of KNXnet/IP Secure, that fixes both theoretical and practical security risks. Future work could include research into side-channel attacks against the KNXnet/IP Secure devices, that may permit to extract cryptographic secrets. It would also be of interest to investigate additional features provided by some devices, such as web interfaces or remote management through a cloud service. Furthermore, Data Secure requires a thorough analysis as well.

References

[1] A. Judmayer, L. Krammer, and W. Kastner, “On the security of security extensions for IP-based KNX networks,” in *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, IEEE, 2014, pp. 1–10. DOI: 10.1109/WFCS.2014.6837593.

[2] D. Whiting, R. Housley, and N. Ferguson, “Submission to NIST: Counter with CBC-MAC (CCM),” 2002. [Online]. Available: https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/ccm/ccm.pdf (visited on 2021-10-08).

[3] M. Dworkin, “Block Cipher Modes of Operation: The CCM Mode For Authentication and Confidentiality,” National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Publication (SP) 800-38C, Updated 2007, May 2004. DOI: 10.6028/NIST.SP.800-38C.

[4] C. Cremers and M. Feltz, “Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal,” in *European Symposium on Research in Computer Security*, Springer, 2012, pp. 734–751. DOI: 10.1007/978-3-642-33167-1_42.

[5] D. Boneh and V. Shoup, *A graduate course in applied cryptography*. Self-publishing, 2020. [Online]. Available: http://toc.cryptobook.us/book.pdf (visited on 2021-10-08).

[6] *Open data communication in building automation, controls and building management — Home and building electronic systems — KNXnet/IP communication*, ISO 22510:2019, International Organization for Standardization, Vernier, Geneva, Switzerland, Nov. 2019.

[7] A. Langley, M. Hamburg, and S. Turner, “Elliptic Curves for Security,” Internet Engineering Task Force (IETF), RFC 7748, Jan. 2016.

[8] K. Moriarty, B. Kaliski, and A. Rusch, “PKCS #5: Password-Based Cryptography Specification Version 2.1,” Internet Engineering Task Force (IETF), RFC 8018, Jan. 2017.

[9] “Secure Hash Standard,” National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST FIPS 180-4, Aug. 2015. DOI: 10.6028/NIST.FIPS.180-4.

[10] “KNX Basics,” KNX Association, Diegem, Brussels, Belgium. [Online]. Available: https://www.knx.org/wAssets/docs/downloads/Marketing/Flyers/KNX-Basics/KNX-Basics_en.pdf (visited on 2021-10-26).

[11] “Journal,” KNX Association, Diegem, Brussels, Belgium, Mar. 9, 2020. [Online]. Available: https://www.knx.org/wAssets/docs/downloads/Marketing/KNX-Journal/International-Journals/English/KNX-Journal-2020_en.pdf (visited on 2020-05-13).

[12] M. Ruta, F. Scioscia, G. Loseto, and E. Di Sciascio, "KNX: A Worldwide Standard Protocol for Home and Building Automation: State of the Art and Perspectives," *Industrial Communication Technology Handbook*, pp. 58–77, 2017.

[13] *KNX Standard*, version 2.1, KNX Association, Diegem, Brussels, Belgium, Dec. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[14] *System Specifications, Communication Media, Twisted Pair 1*, version 01.02.02, KNX Association, Diegem, Brussels, Belgium, Dec. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[15] *System Specifications KNXnet/IP, Overview*, version 01.04.02, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[16] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, "Security in networked building automation systems," in *IEEE International Workshop on Factory Communication Systems*, IEEE, 2006, pp. 283–292. DOI: `10.1109/WFCS.2006.1704168`.

[17] A. Antonini, F. Maggi, and S. Zanero, "A practical attack against a KNX-based building automation system," in *2nd International Symposium for ICS & SCADA Cyber Security Research 2014 (ICS-CSR 2014)*, 2014, pp. 53–60.

[18] J. Molina, "Learn how to control every room at a luxury hotel remotely: The dangers of insecure home automation deployment," *Black Hat USA*, 2014.

[19] *KNXnet/IP Secure*, Application Note 159/13 v04, Draft Proposal, KNX Association, Diegem, Brussels, Belgium, Sep. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[20] "ISO 22510:2019," International Organization for Standardization, [Online]. Available: `https://www.iso.org/standard/73364.html` (visited on 2021-11-20).

[21] "Journal," KNX Association, Diegem, Brussels, Belgium, Mar. 11, 2019. [Online]. Available: `https://www.knx.org/wAssets/docs/downloads/Marketing/KNX-Journal/International-Journals/English/KNX-Journal-2019_en.pdf` (visited on 2020-05-15).

[22] *KNX Data Security*, Application Note 158/13 v02, Draft Proposal, KNX Association, Diegem, Brussels, Belgium, May 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[23] "Journal," KNX Association, Diegem, Brussels, Belgium, Sep. 17, 2018. [Online]. Available: `https://www.knx.org/wAssets/docs/downloads/Marketing/KNX-Journal/International-Journals/English/KNX-Journal-2-2018_en.pdf` (visited on 2021-06-08).

[24] *KNX Certification of Products, Procedure*, version 01.03.01, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[25] W. Granzer, D. Lechner, F. Praus, and W. Kastner, "Securing IP backbones in building automation networks," in *2009 7th IEEE International Conference on Industrial Informatics*, IEEE, 2009, pp. 410–415. DOI: `10.1109/INDIN.2009.5195839`.

[26] H. Glanzer, L. Krammer, and W. Kastner, "Increasing security and availability in KNX networks," in *Sicherheit 2016 - Sicherheit, Schutz und Zuverlässigkeit*, Gesellschaft für Informatik eV., 2016, pp. 241–252.

[27] S. Seifried, G. Gridling, and W. Kastner, "KNX IPv6: Design issues and proposed architecture," in *2017 13th International Workshop on Factory Communication Systems (WFCS)*, IEEE, 2017, pp. 1–10. DOI: `10.1109/WFCS.2017.7991951`.

[28] J. Goltz, "Sicherheitsanalyse von Gebäudeautomationsnetzen auf Feldbusebene am Beispiel von KNX," M.S. thesis, Universität Rostock, 2018.

[29] C. Vacherot, "Sneak into buildings with KNXnet/IP," 2020. HAL: `hal-03022310`.

[30] J. Katz and Y. Lindell, *Introduction to modern cryptography.* CRC press, 2020.

[31] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography.* CRC press, 1996.

[32] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography.* Springer Science & Business Media, 2006.

[33] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2000, pp. 531–545. DOI: `10.1007/3-540-44448-3_41`.

[34] H. Krawczyk, "The order of encryption and authentication for protecting communications (or: How secure is SSL?)" In *Annual International Cryptology Conference*, Springer, 2001, pp. 310–331. DOI: `10.1007/3-540-44647-8_19`.

[35] J. Jonsson, "On the security of CTR+CBC-MAC," in *International Workshop on Selected Areas in Cryptography*, Springer, 2002, pp. 76–93. DOI: `10.1007/3-540-36492-7_7`.

[36] P. Rogaway and D. A. Wagner, "A Critique of CCM," *Cryptology ePrint Archive*, 2003. [Online]. Available: `http://eprint.iacr.org/2003/070` (visited on 2021-10-09).

[37] M. Dworkin, E. Barker, J. Nechvatal, *et al.*, "Advanced Encryption Standard (AES)," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST FIPS 197, Nov. 2001. DOI: `10.6028/NIST.FIPS.197`.

[38] P. Rogaway, "Evaluation of some blockcipher modes of operation," *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan*, 2011.

[39] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques*, Springer, 1985, pp. 417–426. DOI: 10.1007/3-540-39799-X_31.

[40] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987. DOI: 10.1090/S0025-5718-1987-0866109-5.

[41] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," *Designs, codes and cryptography*, vol. 19, no. 2, pp. 173–193, 2000. DOI: 10.1023/A:1008354106356.

[42] D. J. Bernstein, "Curve25519: New Diffie-Hellman speed records," in *International Workshop on Public Key Cryptography*, Springer, 2006, pp. 207–228. DOI: 10.1007/11745853_14.

[43] D. Genkin, L. Valenta, and Y. Yarom, "May the fourth be with you: A microarchitectural side channel attack on several real-world applications of curve25519," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 845–858. DOI: 10.1145/3133956.3134029.

[44] "Kryptographische Verfahren: Empfehlungen und Schlüssellängen," Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, Technical Guideline BSI TR-02102-1, version 2021-01, Mar. 2021.

[45] E. Barker and A. Roginsky, "Transitioning the Use of Cryptographic Algorithms and Key Lengths," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Publication (SP) 800-131A, Rev. 2, May 2019. DOI: 10.6028/NIST.SP.800-131Ar2.

[46] "Elliptic Curve Cryptography," Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, Technical Guideline BSI TR-03111, Jun. 2018.

[47] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976. DOI: 10.1109/TIT.1976.1055638.

[48] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and cryptography*, vol. 2, no. 2, pp. 107–125, 1992. DOI: 10.1007/BF00124891.

[49] *KNXnet/IP Security*, Application Note 159/13 v06, Draft for Voting, KNX Association, Diegem, Brussels, Belgium, Oct. 2017, Unpublished.

[50] S. Blake-Wilson and A. Menezes, “Unknown key-share attacks on the station-to-station (STS) protocol,” in *International Workshop on Public Key Cryptography*, Springer, 1999, pp. 154–170. DOI: 10.1007/3-540-49162-7_12.

[51] K.-K. R. Choo, “Key establishment: Proofs and refutations,” Ph.D. dissertation, Queensland University of Technology, 2006.

[52] B. Lipp, B. Blanchet, and K. Bhargavan, “A mechanised cryptographic proof of the WireGuard virtual private network protocol,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2019, pp. 231–246. DOI: 10.1109/EuroSP.2019.00026.

[53] D. G. Steer, L. Strawczynski, W. Diffie, and M. Wiener, “A secure audio teleconference system,” in *Conference on the Theory and Application of Cryptography*, Springer, 1988, pp. 520–528. DOI: 10.1007/0-387-34799-2_37.

[54] Y. Kim, A. Perrig, and G. Tsudik, “Communication-efficient group key agreement,” in *IFIP International Information Security Conference*, Springer, 2001, pp. 229–244. DOI: 10.1007/0-306-46998-7_16.

[55] M. Burmester and Y. Desmedt, “A secure and efficient conference key distribution system,” in *Workshop on the Theory and Application of of Cryptographic Techniques*, Springer, 1994, pp. 275–286. DOI: 10.1007/BFb0053443.

[56] M. Steiner, G. Tsudik, and M. Waidner, “Key agreement in dynamic peer groups,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, pp. 769–780, 2000. DOI: 10.1109/71.877936.

[57] Y. Kim, A. Perrig, and G. Tsudik, “Simple and fault-tolerant key agreement for dynamic collaborative groups,” in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, 2000, pp. 235–244. DOI: 10.1145/352600.352638.

[58] Y. Kim, A. Perrig, and G. Tsudik, “Tree-based group key agreement,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, pp. 60–96, 2004. DOI: 10.1145/984334.984337.

[59] M. Manulis, “Contributory group key agreement protocols, revisited for mobile ad-hoc groups,” in *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, IEEE, 2005, pp. 811–818. DOI: 10.1109/MAHSS.2005.1542876.

[60] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, “On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1802–1819. DOI: 10.1145/3243734.3243747.

[61] J. Alwen, S. Coretti, Y. Dodis, and Y. Tselekounis, "Security analysis and improvements for the IETF MLS standard for group messaging," in *Annual International Cryptology Conference*, Springer, 2020, pp. 248–277. DOI: 10.1007/978-3-030-56784-2_9.

[62] J. Alwen, S. Coretti, D. Jost, and M. Mularczyk, "Continuous group key agreement with active security," in *Theory of Cryptography Conference*, Springer, 2020, pp. 261–290. DOI: 10.1007/978-3-030-64378-2_10.

[63] J. Alwen, M. Capretto, M. Cueto, *et al.*, "Keep the dirt: Tainted treekem, adaptively and actively secure continuous group key agreement," in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021, pp. 268–284. DOI: 10.1109/SP40001.2021.00035.

[64] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983. DOI: 10.1109/TIT.1983.1056650.

[65] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Annual international cryptology conference*, Springer, 1993, pp. 232–249. DOI: 10.1007/3-540-48329-2_21.

[66] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *International conference on the theory and applications of cryptographic techniques*, Springer, 2001, pp. 453–474. DOI: 10.1007/3-540-44987-6_28.

[67] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *International conference on provable security*, Springer, 2007, pp. 1–16. DOI: 10.1007/978-3-540-75670-5_1.

[68] C. Cremers and M. Feltz, "Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal," version 2.0, *Cryptology ePrint Archive*, Oct. 19, 2017. [Online]. Available: https://eprint.iacr.org/2012/416.pdf (visited on 2021-09-15).

[69] C. M. Swanson, "Security in key agreement: Two-party certificateless schemes," M.S. thesis, University of Waterloo, 2008.

[70] A. Visconti, S. Bossi, H. Ragab, and A. Calò, "On the weaknesses of PBKDF2," in *International Conference on Cryptology and Network Security*, Springer, 2015, pp. 119–126. DOI: 10.1007/978-3-319-26823-1_9.

[71] A. Visconti, O. Mosnáček, M. Brož, and V. Matyáš, "Examining PBKDF2 security margin - Case study of LUKS," *Journal of Information Security and Applications*, vol. 46, pp. 296–306, 2019. DOI: 10.1016/j.jisa.2019.03.016.

[72] H. Choi and S. C. Seo, "Optimization of PBKDF2 Using HMAC-SHA2 and HMAC-LSH Families in CPU Environment," *IEEE Access*, vol. 9, pp. 40 165–40 177, 2021. DOI: 10.1109/ACCESS.2021.3065082.

[73] E. Biham and P. C. Kocher, "A known plaintext attack on the PKZIP stream cipher," in *International Workshop on Fast Software Encryption*, Springer, 1994, pp. 144–153. DOI: 10.1007/3-540-60590-8_12.

[74] M. Stay, "ZIP attacks with reduced known plaintext," in *International Workshop on Fast Software Encryption*, Springer, 2001, pp. 125–134. DOI: 10.1007/3-540-45473-X_10.

[75] R. Cavada, A. Cimatti, M. Dorigatti, *et al.*, "The nuXmv symbolic model checker," in *International Conference on Computer Aided Verification*, Springer, 2014, pp. 334–342. DOI: 10.1007/978-3-319-08867-9_22.

[76] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, IEEE, 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[77] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Workshop on Logic of Programs*, Springer, 1981, pp. 52–71. DOI: 10.1007/BFb0025774.

[78] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986. DOI: 10.1109/TC.1986.1676819.

[79] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *International conference on tools and algorithms for the construction and analysis of systems*, Springer, 1999, pp. 193–207. DOI: 10.1007/3-540-49059-0_14.

[80] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman, "Completeness and complexity of bounded model checking," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer, 2004, pp. 85–96. DOI: 10.1007/978-3-540-24622-0_9.

[81] J. De Ruiter and E. Poll, "Protocol State Fuzzing of TLS Implementations," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 193–206.

[82] J. De Ruiter. "StateLearner," [Online]. Available: https://github.com/jderuiter/statelearner (visited on 2021-11-20).

[83] M. Isberner, F. Howar, and B. Steffen, "The open-source learnlib," in *International Conference on Computer Aided Verification*, Springer, 2015, pp. 487–495. DOI: 10.1007/978-3-319-21690-4_32.

[84] P. Fiterău-Broştean, B. Jonsson, R. Merget, J. de Ruiter, K. Sagonas, and J. Somorovsky, "Analysis of DTLS Implementations Using Protocol State Fuzzing," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2523–2540.

[85] *BSI-Standard 200-3, Risik Analysis based on IT-Grundschutz*, version 1.0, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, May 2018.

[86] *BSI-Standard 200-3, Risikomanagement*, version 1.0, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, Nov. 2017.

[87] *BSI-Standard 200-1, Information Security Management Systems (ISMS)*, version 1.0, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, May 2018.

[88] *BSI-Standard 200-1, Managementsysteme für Informationssicherheit (ISMS)*, version 1.0, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, Nov. 2017.

[89] *BSI-Standard 200-2, IT-Grundschutz Methodology*, version 1.0, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, May 2018.

[90] *BSI-Standard 200-2, IT-Grundschutz-Methodik*, version 1.0, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, Nov. 2017.

[91] *IT-Grundschutz-Kompendium*, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, Feb. 2021.

[92] *Sichere Anbindung von lokalen Netzen an das Internet (ISi-LANA), BSI-Standards zur Internet-Sicherheit (ISi-S)*, version 2.1, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, Aug. 2014.

[93] *Sicherer Fernzugriff auf das interne Netz (ISi-Fern), BSI-Studie zur Internet-Sicherheit (ISi-S)*, version 1.2, Bundesamt für Sicherheit in der Informationstechnik, Bonn, North Rhine-Westphalia, Germany, May 2021.

[94] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Publication (SP) 800-207, 2020. DOI: `10.6028/NIST.SP.800-207`.

[95] "Zero Trust Maturity Model," Cybersecurity and Infrastructure Security Agency, Rosslyn, VA, USA, Tech. Rep. Pre-decisional Draft, version 1.0, May 2020.

[96] *KNX System Specifications, Architecture*, version 03.00.02, KNX Association, Diegem, Brussels, Belgium, Nov. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[97] *System Specifications, Communication, Data Link Layer General*, version 01.02.02, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[98] *System Specifications, Communication Media, KNX IP*, version 01.00.01, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[99] *System Specifications KNXnet/IP, Core*, version 01.05.01, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[100] *System Specifications, Communication, Application Layer*, version 01.06.02, KNX Association, Diegem, Brussels, Belgium, Dec. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[101] *System Specifications, Standardised Interfaces, External Message Interface*, version 01.03.03, KNX Association, Diegem, Brussels, Belgium, Nov. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[102] *System Specifications, Communication Media, Powerline*, version 02.02.02, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[103] *System Specifications, Communication Media, Radio Frequency*, version 01.06.03, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[104] *System Specifications, Communication, Network Layer*, version 01.01.02, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[105] *System Specifications, Communication, Transport Layer*, version 01.02.02, KNX Association, Diegem, Brussels, Belgium, Nov. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[106] *System Specifications, Management, Resources*, version 01.09.03, KNX Association, Diegem, Brussels, Belgium, Dec. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[107] *System Specifications KNXnet/IP, Tunnelling*, version 01.05.03, KNX Association, Diegem, Brussels, Belgium, Oct. 2013. [Online]. Available: `https://my.knx.org/de/shop/knx-specifications` (visited on 2020-05-02).

[108] D. J. Bernstein and T. Lange, “Montgomery curves and the montgomery ladder.,” *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 293, 2017.

[109] P. L. Montgomery, “Speeding the pollard and elliptic curve methods of factorization,” *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987. DOI: `10.1090/S0025-5718-1987-0866113-7`.

[110] V. Shoup, “Lower bounds for discrete logarithms and related problems,” in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1997, pp. 256–266. DOI: `10.1007/3-540-69053-0_18`.

[111] S. D. Galbraith and P. Gaudry, “Recent progress on the elliptic curve discrete logarithm problem,” *Designs, Codes and Cryptography*, vol. 78, no. 1, pp. 51–72, 2016. DOI: `10.1007/s10623-015-0146-7`.

[112] C. Cremers and D. Jackson, "Prime, order please! revisiting small subgroup and invalid curve attacks on protocols using diffie-hellman," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, IEEE, 2019, pp. 78–7815. DOI: 10.1109/CSF.2019.00013.

[113] J.-P. Aumasson. "Should Curve25519 keys be validated?" (Apr. 2017), [Online]. Available: https://research.kudelskisecurity.com/2017/04/25/should-ecdh-keys-be-validated/ (visited on 2022-02-23).

[114] T. Perrin. "X25519 and zero outputs." (May 2017), [Online]. Available: https://moderncrypto.org/mail-archive/curves/2017/000896.html (visited on 2022-02-23).

[115] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," in *Annual International Cryptology Conference*, Springer, 2003, pp. 617–630. DOI: 10.1007/978-3-540-45146-4_36.

[116] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Annual international cryptology conference*, Springer, 1996, pp. 1–15. DOI: 10.1007/3-540-68697-5_1.

[117] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," Internet Engineering Task Force (IETF), RFC 2104, Feb. 1997.

[118] M. S. Turan, E. Barker, W. Burr, and L. Chen, "Recommendation for password-based key derivation: Part 1: Storage Applications," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Publication (SP) 800-132, 2010. DOI: 10.6028/NIST.SP.800-132.

[119] S. Blake-Wilson and A. Menezes, "Authenticated Diffe-Hellman key agreement protocols," in *International Workshop on Selected Areas in Cryptography*, Springer, 1998, pp. 339–361. DOI: 10.1007/3-540-48892-8_26.

[120] C. Cremers, "Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2," in *European Symposium on Research in Computer Security*, Springer, 2011, pp. 315–334. DOI: 10.1007/978-3-642-23822-2_18.

[121] J. Gill, "Computational complexity of probabilistic turing machines," *SIAM Journal on Computing*, vol. 6, no. 4, pp. 675–695, 1977. DOI: 10.1137/0206049.

[122] *KNX Data Security*, Application Note 158/13 v04, Approved Standard, KNX Association, Diegem, Brussels, Belgium, Jan. 2018, Unpublished.

[123] M. Bozzano, R. Cavada, A. Cimatti, *et al.*, *nuXmv 2.0.0 User Manual*, Fondazione Bruno Kessler, Provo, Trento, Italy, 2019.

[124] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987. DOI: 10.1016/0890-5401(87)90052-6.

[125] T. S. Chow, “Testing software design modeled by finite-state machines,” *IEEE transactions on software engineering*, no. 3, pp. 178–187, 1978. DOI: 10.1109/TSE.1978.231496.

[126] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis, “Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography,” National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Publication (SP) 800-56A, Rev. 3, Apr. 2018. DOI: 10.6028/NIST.SP.800-56Ar3.

[127] J. A. Donenfeld, “Wireguard: Next generation kernel network tunnel.,” in *24th Annual Network and Distributed System Security Symposium (NDSS)*, 2017, pp. 1–12.

[128] M. Marlinspike and T. Perrin. “The Double Ratchet Algorithm.” (Nov. 2016), [Online]. Available: https://www.signal.org/docs/specifications/doubleratchet/doubleratchet.pdf (visited on 2022-02-23).

[129] C. Percival, *Stronger key derivation via sequential memory-hard functions*, 2009.

[130] A. Biryukov, D. Dinu, and D. Khovratovich, “Argon2: New generation of memory-hard functions for password hashing and other applications,” in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2016, pp. 292–302. DOI: 10.1109/EuroSP.2016.31.

[131] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson, “Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications,” Internet Research Task Force (IRTF), RFC 9106, Sep. 2021.

[132] H. Krawczyk, “Cryptographic extraction and key derivation: The hkdf scheme,” in *Annual Cryptology Conference*, Springer, 2010, pp. 631–648. DOI: 10.1007/978-3-642-14623-7_34.

[133] “hashcat - advanced password recovery,” [Online]. Available: https://hashcat.net/hashcat/ (visited on 2022-03-21).

[134] “CWE-760: Use of a One-Way Hash with a Predictable Salt,” The MITRE Corporation, [Online]. Available: https://cwe.mitre.org/data/definitions/760.html (visited on 2021-02-27).

[135] “CWE-798: Use of Hard-coded Credentials,” The MITRE Corporation, [Online]. Available: https://cwe.mitre.org/data/definitions/798.html (visited on 2021-02-27).

[136] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, “High-speed high-security signatures,” *Journal of cryptographic engineering*, vol. 2, no. 2, pp. 77–89, 2012. DOI: 10.1007/s13389-012-0027-1.

[137] K. Cohn-Gordon, C. Cremers, and L. Garratt, “On post-compromise security,” in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, IEEE, 2016, pp. 164–178. DOI: 10.1109/CSF.2016.19.

[138] V. Cerf, “ASCII format for Network Interchange,” Network Working Group, RFC 20, Oct. 1969.

[139] “GeForce GTX 10-Serie,” [Online]. Available: `https://www.nvidia.com/de-de/geforce/10-series/` (visited on 2022-03-21).

[140] “Stromkosten der energieintensiven Industrie,” Frauenhofe ISI and ECOFYS, Jun. 2015. [Online]. Available: `https://www.isi.fraunhofer.de/content/dam/isi/dokumente/ccx/2015/Industriestrompreise_Abschlussbericht.pdf` (visited on 2022-03-22).

[141] “Bericht - Monitoringbericht 2020,” Bundesnetzagentur, Mar. 2021. [Online]. Available: `https://www.bundesnetzagentur.de/SharedDocs/Mediathek/Monitoringberichte/Monitoringbericht_Energie2020.pdf` (visited on 2022-03-22).

[142] M. Sauter and S. Wochnik. “Der neue Ti-tan.” (Mar. 9, 2017), [Online]. Available: `https://www.golem.de/news/geforce-gtx-1080-ti-im-test-der-neue-ti-tan-1703-126531-3.html` (visited on 2022-03-21).

[143] V. Lourdas, “Project Password,” KNX.org, Oct. 4, 2021. [Online]. Available: `https://web.archive.org/web/20211021073825/https://support.knx.org/hc/en-us/articles/360011660999` (visited on 2021-10-27).

[144] S. Josefsson, “The Base16, Base32, and Base64 Data Encodings,” Network Working Group, RFC 4648, Oct. 2006.

[145] “ILSpy,” [Online]. Available: `https://github.com/icsharpcode/ILSpy` (visited on 2022-03-21).

[146] “de4dot,” [Online]. Available: `https://github.com/de4dot/de4dot` (visited on 2022-03-21).

[147] “Reference Source .NET Framework 4.8,” [Online]. Available: `https://referencesource.microsoft.com/#mscorlib/system/security/cryptography/passwordderivebytes.cs` (visited on 2021-07-18).

[148] B. Kaliski, “PKCS #7: Cryptographic Message Syntax,” Network Working Group, RFC 2315, Mar. 1998.

[149] “CWE-321: Use of Hard-coded Cryptographic Key,” The MITRE Corporation, [Online]. Available: `https://cwe.mitre.org/data/definitions/321.html` (visited on 2021-02-27).

[150] “ProgramData,” [Online]. Available: `https://docs.microsoft.com/en-us/windows-hardware/customize/desktop/unattend/microsoft-windows-shell-setup-folderlocations-programdata` (visited on 2022-03-23).

[151] J. Demarest, KNX Association, personal correspondence, Jul. 2021.

[152] V. Lourdas, KNX Association, personal correspondence, Nov. 2021.

[153] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, “PKCS #1: RSA Cryptography Specifications Version 2.2,” Internet Engineering Task Force (IETF), RFC 8017, Nov. 2016.

[154] G. Leurent and T. Peyrin, “From collisions to chosen-prefix collisions application to full sha-1,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2019, pp. 527–555. DOI: 10.1007/978-3-030-17659-4_18.

[155] G. Leurent and T. Peyrin, “SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1839–1856.

[156] U. Unal. “Certification introduction,” [Online]. Available: https://support.knx.org/hc/en-us/articles/360000041039-Certification-introduction (visited on 2022-03-23).

[157] “KNX Project Preparation,” KNX Association, Diegem, Brussels, Belgium, version 11.19. [Online]. Available: https://www.knx.org/wAssets/docs/downloads/Marketing/Flyers/KNX-Project-Preparation/KNX-Project-Preparation_en.pdf (visited on 2021-06-08).

[158] M. Antonakakis, T. April, M. Bailey, *et al.*, “Understanding the mirai botnet,” in *26th USENIX security symposium (USENIX Security 17)*, 2017, pp. 1093–1110.

[159] “Fernwartung des KNX-Systems,” Albrecht JUNG GmbH & Co. KG, [Online]. Available: https://www.jung.de/10326/produkte/technik/knx-system/ips-remote/ (visited on 2021-11-28).

[160] “Gira S1,” Giersiepen GmbH & Co. KG, [Online]. Available: https://partner.gira.de/systeme/knx-system/knx-produkte/systemgeraete/s1/features.html (visited on 2021-11-28).

[161] “Journal,” KNX Association, Diegem, Brussels, Belgium, Jan. 21, 2021. [Online]. Available: https://www.knx.org/wAssets/docs/downloads/Marketing/KNX-Journal/International-Journals/English/KNX-Journal-2021_en.pdf (visited on 2021-02-05).

A. Appendix

A.1. Supplementary Analysis with eCK-PFS

Attack Game A.1 (**Password reuse**). Let attack game G in model eCK-PFS be played by a PPT adversary $\mathcal{A}$ under the assumption that there exist at least two servers $\hat{S}_i, \hat{S}_j \in \mathcal{P}$ and a $pwd' = (k_{pwd}, uid)$ for which $pwd' \in PWD_{\hat{S}_i}$ and $pwd' \in PWD_{\hat{S}_j}$. Let $U = \{\hat{S}_1, ..., \hat{S}_n\}$ be the set of uncorrupted servers in $\mathcal{P}$ and $V = \{\}$ the set of corrupted servers.

1. For each $\hat{S}_i \in U$ do:
   a) $\mathcal{A}$ issues corrupt($\hat{S}_i$). It reveals $PWD_{\hat{S}_i}$ and $DAC_{\hat{S}_i}$. Update $U = U \backslash \hat{S}_i$ and $Q = Q \cup \hat{S}_i$.
   b) For each $\hat{S}_j \in U$ do:
      i. For each $pwd'_l \in PWD_{\hat{S}}$ do:
         A. $\mathcal{A}$ issues send($s, \hat{S}_j$) to start an initiator session s with $\hat{S}_j$, where $s_{peer} = \hat{S}_j$ and $s_{role} = \mathcal{I}$. It computes $esk_{\mathcal{A}} \xleftarrow{R} \{0, 1, ..., 2^{256} - 1\}$ and $epk_{\mathcal{A}} \leftarrow$ X25519($esk_{\mathcal{A}}, 9$).
         B. $\mathcal{A}$ issues send(s, m), where m is a `SESSION_REQUEST` containing $epk_{\mathcal{A}}$. The $\hat{S}_j$ computes $esk_{\hat{S}_j} \xleftarrow{R} \{0, 1, ..., 2^{256} - 1\}$, $epk_{\hat{S}_j} \leftarrow$ X25519($esk_{\hat{S}_j}, 9$), $k_s \leftarrow$ MSB$_{128}$(SHA-256(X25519($esk_{\hat{S}_j}, epk_{\mathcal{A}}$))) on reception. It replies with a `SESSION_RESPONSE` frame containing $epk_{\hat{S}_j}$. The k_{dac} for $\hat{S}_j$ is used as key for CCM.
         C. $\mathcal{A}$ receives the frame and uses the contained $epk_{\hat{S}_j}$ to compute $k_s \leftarrow$ MSB$_{128}$(SHA-256(X25519($esk_{\mathcal{A}}, epk_{\hat{S}_j}$))). It issues a send($s, m$), where m is a `SESSION_AUTHENTICATE` encapsulated in a `SECURE_-WRAPPER`. The `SESSION_AUTHENTICATE` contains the $uid \in pwd'_l$ and $k_{pwd} \in pwd'_l$ is used as key for CCM. The `SECURE_WRAPPER` is encrypted and authenticated with k_s.
         D. $\hat{S}_j$ receives and decrypt the `SECURE_WRAPPER` with k_s. The uid contained in the frame identifies a $pwd' \in PWD_{\hat{S}_j}$ with $pwd' = (k_{pwd}, uid)$. It is assumed that the uid uniquely determines a tuple, since it is originally the array index of `PID_PASSWORD_HASHES`. If $\hat{S}_j$ is able to verify the MAC of `SESSION_AUTHENTICATE` using k_{pwd}, then $pwd' = pwd'_l$. It sends a `SESSION_STATUS` indicating success, encapsulated in a `SECURE_WRAPPER` that is encrypted and authenticated with k_s.
         E. If $\mathcal{A}$ receives a positive response, the session is completed. Break out of all loop and go to step 2. Otherwise, continue.
2. The last created session s is fresh, because:
   a) G does not include the query session-key(s).

b) There is no session-key(s^*) query issued for any session s^* matching s.

c) G does not include the ephemeral-key(s) query.

d) For no origin session s' to session s does G include a corrupt(s_{peer}) and ephemeral-key(s') query.

e) G does not use corrupt(s_{peer}) and there is an origin session s' to s, namely the one of the last server $\hat{S}_j$.

$\mathcal{A}$ issues test-session(s). The challenger provides either the real session key or a random session key. $\mathcal{A}$ can determine with certainty which key they have been given, since they know the session key k_s. Thus, b' is selected accordingly and $P(b = b') = 1$. $\mathcal{A}$ wins G with $\mathrm{Adv}_G^{\text{eCK-PFS}}(\lambda) = 1$. The KNXnet/IP Secure unicast protocol is not secure in eCK-PFS, because no negligible function negl(λ) exists such that $\mathrm{Adv}_G^{\text{eCK-PFS}}(\lambda) \leq \mathrm{negl}(\lambda)$. This is evidently the case, because any negl(λ) would have to fulfill definition 3.34 and $\mathrm{Adv}_G^{\text{eCK-PFS}}(\lambda)$ is a constant non-zero value.

Attack Game A.2 (**Skipped server authentication**)**.** Let attack game G in model eCK-PFS be played by a PPT adversary $\mathcal{A}$ under the assumption that at least one client $\hat{C}_i \in \mathcal{P}$ skips the server authentication.

1. For each $\hat{C}_i \in \mathcal{P}$ do:

   a) $\mathcal{A}$ prevents the `SESSION_REQUEST` of a $\hat{C}_i$ from reaching a $\hat{S}_j \in \mathcal{P}$, with which it initiated a session s, where $s_{actor} = \hat{C}_i$, $s_{peer} = \hat{S}_j$ and $s_{role} = \mathcal{I}$. $\mathcal{A}$ computes $esk_{\mathcal{A}} \xleftarrow{R} \{0, 1, ..., 2^{256} - 1\}$ and $epk_{\mathcal{A}} \leftarrow \mathrm{X25519}(esk_{\mathcal{A}}, 9)$. With the $epk_{\hat{C}}$ contained in the `SESSION_REQUEST`, $\mathcal{A}$ determines the session key with $k_s \leftarrow \mathrm{MSB}_{128}(\text{SHA-256}(\mathrm{X25519}(esk_{\mathcal{A}}, epk_{\hat{C}})))$.

   b) $\mathcal{A}$ uses send($s, \hat{S}, m$) to start the responder session s' to s on behalf of $\hat{S}$, where m is a `SESSION_RESPONSE` frame, containing $epk_{\mathcal{A}}$. The MAC field is set to an arbitrary incorrect value.

   c) If $\hat{C}_i$ validates the MAC, the authentication fails and the session is terminated. If $\hat{C}_i$ skips the server authentication, then it will compute $k_s \leftarrow \mathrm{MSB}_{128}(\text{SHA-256}(\mathrm{X25519}(esk_{\hat{C}_2}, epk_{\mathcal{A}})))$ and a `SESSION_AUTHENTICATE` frame is sent as a reply, which is encapsulated in a `SECURE_WRAPPER` that is encrypted and authenticated with k_s.

   d) If the session has not been terminated, $\mathcal{A}$ issues send(s, m) where m is a `SESSION_STATUS` indicating successful authentication, which is encapsulated in a `SECURE_WRAPPER` that is encrypted and authenticated with k_s. The session s is completed. Break out of the loop and continue with step 2. If the session has been terminated, continue with the next client.

2. The session s is fresh because:

   a) G does not include the query session-key(s).

   b) There is no session-key(s^*) query issued for any session s^* matching s.

c) G does not include the ephemeral-key(s) query.

d) For no origin session s' to session s does G include a corrupt(s_{peer}) and ephemeral-key(s') query.

e) G does not use corrupt(s_{peer}) and there is an origin session s' to s, namely the one created by the adversary.

$\mathcal{A}$ issues test-session(s). The challenger provides either the real session key or a random session key. $\mathcal{A}$ can determine with certainty which key they have been given, since they know the session key k_s. Thus, b' is selected accordingly and $P(b = b') = 1$. $\mathcal{A}$ wins G with $\mathrm{Adv}_G^{\text{eCK-PFS}}(\lambda) = 1$. The KNXnet/IP Secure unicast protocol is not secure in eCK-PFS, because no negligible function negl(λ) exists such that $\mathrm{Adv}_G^{\text{eCK-PFS}}(\lambda) \leq \mathrm{negl}(\lambda)$. This is evidently the case, because any negl(λ) would have to fulfill definition 3.34 and $\mathrm{Adv}_G^{\text{eCK-PFS}}(\lambda)$ is a constant non-zero value.

A.2. Model Checking

No.	Target Models	CTL
1.	Session FSM, Session FSM with Timer CTL based on [6, p. 132]	`AG (((fsm.sent_msg = Close | fsm.sent_msg = AuthenticationFailed) | fsm.sent_msg = Unauthenticated) -> fsm._state_ = IDLE)`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 132]	`AG (((fsm.sent_msg = Close | fsm.sent_msg = AuthenticationFailed) | fsm.sent_msg = Unauthenticated) -> fsm._state_ = 0)`
2.	Session FSM, Session FSM with Timer CTL based on [6, p. 144]	`AG !((fsm._state_ = IDLE & fsm.recv_msg != SessionRequest) & EX fsm._state_ != IDLE)`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 144]	`AG !((fsm._state_ = 0 & fsm.recv_msg != SessionRequest) & EX fsm._state_ != 0)`
3.	Session FSM, Session FSM with Timer CTL based on [6, p. 144]	`(AG (fsm._state_ = IDLE & fsm.recv_msg = SessionRequest) -> AX fsm.sent_msg = SessionResponse)`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 144]	`(AG (fsm._state_ = 0 & fsm.recv_msg = SessionRequest) -> AX fsm.sent_msg = SessionResponse)`
4.	Session FSM with Timer CTL based on [6, p. 144]	`AG ((fsm._state_ = IDLE & recv_msg = SessionRequest) -> AX (fsm._state_ = UNAUTHENTICATED & fsm.session_timer = 10))`

No.	Target Models	CTL
5.	Session FSM with Timer CTL based on [6, p. 144]	`AG ((fsm._state_ = UNAUTHENTICATED & fsm.session_timer = 0) -> AX fsm._state_ = IDLE)`
6.	Session FSM, Session FSM with Timer CTL based on [6, p. 144]	`AG ((fsm._state_ = UNAUTHENTICATED & ((((((fsm.recv_msg = AuthenticationSuccess | fsm.recv_msg = AuthenticationFailed) | fsm.recv_msg = Unauthenticated) | fsm.recv_msg = KeepAlive) | fsm.recv_msg = Timeout) | fsm.recv_msg = Close) | fsm.recv_msg = WrappedFrame)) -> AX fsm.sent_msg = Unauthenticated)`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 144]	`AG ((fsm._state_ = 1 & ((((((fsm.recv_msg = AuthenticationSuccess | fsm.recv_msg = AuthenticationFailed) | fsm.recv_msg = Unauthenticated) | fsm.recv_msg = KeepAlive) | fsm.recv_msg = Timeout) | fsm.recv_msg = Close) | fsm.recv_msg = WrappedFrame)) -> AX fsm.sent_msg = Unauthenticated)`
7.	Session FSM, Session FSM with Timer CTL based on [6, p. 144]	`AG ((fsm._state_ = Unauthenticated & ((recv_msg = InvalidSessionAuthenticateReservedUserID | recv_msg = InvalidSessionAuthenticateUnusedUserID) | recv_msg = InvalidSessionAuthenticateMac)) -> AX fsm.sent_msg = AuthenticationFailed)`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 144]	`AG ((fsm._state_ = 1 & ((recv_msg = InvalidSessionAuthenticateReservedUserID | recv_msg = InvalidSessionAuthenticateUnusedUserID) | recv_msg = InvalidSessionAuthenticateMac)) -> AX fsm.sent_msg = AuthenticationFailed)`
8.	Session FSM CTL based on [6, p. 145]	`AG (((fsm._state_ = AUTHENTICATED & recv_msg = None) & timer_state = expired) -> AX (fsm.sent_msg = Timeout & fsm._state_ = IDLE))`
	Session FSM with Timer CTL based on [6, p. 145]	`AG (((fsm._state_ = AUTHENTICATED & recv_msg = None) & fsm.session_timer = 0) -> AX (fsm.sent_msg = Timeout & fsm._state_ = IDLE))`
9.	Session FSM, Session FSM with Timer CTL based on [6, p. 145]	`AG (((fsm.recv_msg = WrappedFrame & fsm._state_ != AUTHENTICATED) & fsm._state_ != IDLE) -> AX fsm.sent_msg = Unauthenticated)`

No.	Target Models	CTL
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 145]	`AG (((fsm.recv_msg = WrappedFrame & fsm._state_ != 2) & fsm._state_ != 0) -> AX fsm.sent_msg = Unauthenticated)`
10.	Session FSM, Session FSM with Timer CTL based on [6, p. 145]	`AG (fsm.recv_msg = InvalidSecureWrapperSessionIdentifier -> AX (fsm.sent_msg = None | fsm.sent_msg = Timeout))`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 145]	`AG (fsm.recv_msg = InvalidSecureWrapperSessionIdentifier -> AX ((fsm.sent_msg = None | fsm.sent_msg = Timeout) & ((fsm.status = None | fsm.status = ReadTimeout) | fsm.status = ConnectionClosed)))`
11.	Session FSM, Session FSM with Timer CTL based on [6, p. 145]	`AG ((fsm._state_ = AUTHENTICATED & recv_msg = InvalidSecureWrapperMac) -> AX (fsm.sent_msg = None | fsm.sent_msg = Timeout))`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 145]	`AG ((fsm._state_ = 2 & recv_msg = InvalidSecureWrapperMac) -> AX ((fsm.sent_msg = None | fsm.sent_msg = Timeout) & ((fsm.status = None | fsm.status = ReadTimeout) | fsm.status = ConnectionClosed)))`
12.	Session FSM, Session FSM with Timer CTL based on [6, p. 145]	`AG (fsm.recv_msg = Close -> AX fsm._state_ = IDLE)`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 145]	`AG (fsm.recv_msg = Close -> AX fsm._state_ = 0)`
13.	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 145]	`AG (fsm.status = ConnectionClosed -> fsm._state_ = 0)`
14.	Session FSM, Session FSM with Timer CTL based on [6, p. 41]	`AG (fsm.recv_msg = InvalidHeaderVersion -> AX fsm._state_ = IDLE)`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 41]	`AG (fsm.recv_msg = InvalidHeaderVersion -> AX (fsm.status = ConnectionClosed & fsm._state_ = 0))`
15.	Session FSM, Session FSM with Timer CTL based on [6, p. 42]	`AG (((fsm.recv_msg = InvalidHeaderServiceType & fsm._state_ != IDLE) & fsm.timer_state != expired) -> !(EX fsm._state_ = IDLE))`

No.	Target Models	CTL
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 42]	`AG ((fsm.recv_msg = InvalidHeaderServiceType & fsm._state_ != 0) -> !(EX (fsm.status = ConnectionClosed & fsm._state_ = 0)))`
16.	Session FSM, Session FSM with Timer CTL based on [6, p. 122]	`AG (fsm.recv_msg = InvalidSecureWrapperSequenceInformation -> AX (fsm.sent_msg = None | fsm.sent_msg = Timeout))`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 122]	`AG (fsm.recv_msg = InvalidSecureWrapperSequenceInformation -> AX ((fsm.sent_msg = None | fsm.sent_msg = Timeout) & ((fsm.status = None | fsm.status = ReadTimeout) | fsm.status = ConnectionClosed)))`
17.	Session FSM, Session FSM with Timer CTL based on [6, p. 126]	`AG (fsm.recv_msg = InvalidSessionRequestHpaiIp -> AX (fsm.sent_msg = None | fsm.sent_msg = Timeout))`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 126]	`AG (fsm.recv_msg = InvalidSessionRequestHpaiIp -> AX ((fsm.sent_msg = None | fsm.sent_msg = Timeout) & ((fsm.status = None | fsm.status = ReadTimeout) | fsm.status = ConnectionClosed)))`
18.	Session FSM, Session FSM with Timer CTL based on [6, p. 126]	`AG (fsm.recv_msg = InvalidHeaderLength -> AX (fsm.sent_msg = None | fsm.sent_msg = Timeout))`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 126]	`AG (fsm.recv_msg = InvalidHeaderLength -> AX ((fsm.sent_msg = None | fsm.sent_msg = Timeout) & ((fsm.status = None | fsm.status = ReadTimeout) | fsm.status = ConnectionClosed)))`
19.	Session FSM, Session FSM with Timer CTL based on [6, p. 129]	`AG (fsm.recv_msg = InvalidSessionAuthenticateReservedField -> AX (fsm.sent_msg = None | fsm.sent_msg = Timeout))`
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 129]	`AG (fsm.recv_msg = InvalidSessionAuthenticateReservedField -> AX ((fsm.sent_msg = None | fsm.sent_msg = Timeout) & ((fsm.status = None | fsm.status = ReadTimeout) | fsm.status = ConnectionClosed)))`
20.	Session FSM, Session FSM with Timer CTL based on [6, p. 131]	`AG ((fsm._state_ = UNAUTHENTICATED & fsm.recv_msg = SessionAuthenticate) -> AX fsm._state_ = AUTHENTICATED)`

No.	Target Models	CTL
	SCN-IP100.03, KNX IP Router 752 Secure CTL based on [6, p. 131]	`AG ((fsm._state_ = 1 & fsm.recv_msg = SessionAuthenticate) -> AX fsm._state_ = 2)`

21.	Session FSM,
	Session FSM with Timer
		`EF _state_ = IDLE`
		`EF _state_ = UNAUTHENTICATED`
		`EF _state_ = AUTHENTICATED`

	SCN-IP100.03	`EF _state_ = 0`
		`EF _state_ = 1`
		`EF _state_ = 2`
		`EF _state_ = 3`
		`EF _state_ = 4`
		`EF _state_ = 5`

	KNX IP Router 752 Secure	`EF _state_ = 0`
		`EF _state_ = 1`
		`EF _state_ = 2`
		`EF _state_ = 3`
		`EF _state_ = 4`
		`EF _state_ = 5`
		`EF _state_ = 6`
		`EF _state_ = 7`

22.	Session FSM, Session FSM with Timer, SCN-IP100.03, KNX IP Router 752 Secure	`AG !nondeterministic`

23.	Session FSM, Session FSM with Timer	`EF transition_active = IDLE_E00`
		`EF transition_active = UNAUTHENTICATED_E02`
		`EF transition_active = UNAUTHENTICATED_E03`
		`EF transition_active = UNAUTHENTICATED_E04`
		`EF transition_active = UNAUTHENTICATED_E05`
		`EF transition_active = UNAUTHENTICATED_E06`
		`EF transition_active = AUTHENTICATED_None`
		`EF transition_active = AUTHENTICATED_E02`
		`EF transition_active = AUTHENTICATED_E03`
		`EF transition_active = AUTHENTICATED_E04`
		`EF transition_active = AUTHENTICATED_E05`
		`EF transition_active = AUTHENTICATED_E06`

No.	Target Models	CTL
	SCN-IP100.03	`EF transition_active = S0_E31; EF transition_active = S0_E1;`
		`EF transition_active = S0_E34; EF transition_active = S0_E3;`
		`EF transition_active = S1_E4; EF transition_active = S1_E29;`
		`EF transition_active = S1_E30; EF transition_active = S1_E31;`
		`EF transition_active = S1_E34; EF transition_active = S1_E33;`
		`EF transition_active = S1_E10; EF transition_active = S1_E11;`
		`EF transition_active = S2_E12; EF transition_active = S2_E29;`
		`EF transition_active = S2_E31; EF transition_active = S2_E15;`
		`EF transition_active = S2_E16; EF transition_active = S2_E34;`
		`EF transition_active = S3_E18; EF transition_active = S3_E29;`
		`EF transition_active = S3_E30; EF transition_active = S3_E31;`
		`EF transition_active = S3_E22; EF transition_active = S3_E33;`
		`EF transition_active = S3_E34; EF transition_active = S4_E31;`
		`EF transition_active = S4_E26; EF transition_active = S4_E27;`
		`EF transition_active = S5_E28; EF transition_active = S5_E29;`
		`EF transition_active = S5_E30; EF transition_active = S5_E31;`
		`EF transition_active = S5_E32; EF transition_active = S5_E33;`
		`EF transition_active = S5_E34;`
	KNX IP Router 752 Secure	`EF transition_active = S0_E0; EF transition_active = S0_E1;`
		`EF transition_active = S0_E2; EF transition_active = S1_E3;`
		`EF transition_active = S1_E1; EF transition_active = S1_E4;`
		`EF transition_active = S1_E5; EF transition_active = S1_E6;`
		`EF transition_active = S1_E7; EF transition_active = S1_E8;`
		`EF transition_active = S2_E9; EF transition_active = S2_E3;`
		`EF transition_active = S2_E10; EF transition_active = S2_E11;`
		`EF transition_active = S2_E1; EF transition_active = S3_E3;`
		`EF transition_active = S3_E12; EF transition_active = S3_E1;`
		`EF transition_active = S3_E13; EF transition_active = S3_E14;`
		`EF transition_active = S3_E15; EF transition_active = S4_E3;`
		`EF transition_active = S4_E16; EF transition_active = S4_E4;`
		`EF transition_active = S4_E13; EF transition_active = S4_E6;`
		`EF transition_active = S4_E1; EF transition_active = S5_E13;`
		`EF transition_active = S5_E3; EF transition_active = S5_E6;`
		`EF transition_active = S5_E17; EF transition_active = S5_E4;`
		`EF transition_active = S5_E1; EF transition_active = S6_E18;`
		`EF transition_active = S6_E19; EF transition_active = S7_E20;`

Table 15: Tests for the session FSMs

No.	Target Models	CTL/LTL
1.	Timer Sync FSM Integer Clock, with helper variables and wrap around, CTL based on [6, p. 111], adjusted to address non-determinism	`AG (((((received_timer_value > fsm.mc_timer & recv_msg != None) & fsm.notify_timer - time_passed > 0) & !has_joined_new_domain) & !has_sent_secure_wrapper) -> AX fsm.mc_timer = fsm.last_received_timer_value)`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 111], adjusted to address non-determinism	`G (((((received_timer_value > fsm.mc_timer & recv_msg != None) & fsm.notify_timer - time_passed > 0) & !has_joined_new_domain) & !has_sent_secure_wrapper) -> X fsm.mc_timer = fsm.last_received_timer_value)`
2.	Timer Sync FSM Integer Clock, with wrap around, CTL based on [6, p. 111]	`AG ((received_timer_value <= fsm.mc_timer - latencyTolerance) -> AX (!fsm.accepted_frame))`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 111]	`G ((received_timer_value <= fsm.mc_timer - latencyTolerance) -> X (!fsm.accepted_frame))`
3.	Timer Sync FSM Integer Clock, with helper variables and wrap around, CTL based on [6, p. 112], adjusted to address non-determinism	`AG ((!has_joined_new_domain & !has_new_backbone_key) -> AX fsm.mc_timer >= fsm.last_mc_timer)`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 112], adjusted to address non-determinism	`G ((!has_joined_new_domain & !has_new_backbone_key) -> X fsm.mc_timer >= fsm.last_mc_timer)`
4.	Timer Sync FSM Integer Clock, with helper variables and no wrap around, CTL based on [6, p. 112], adjusted to address non-determinism	`AG ((!has_joined_new_domain & !has_new_backbone_key) -> AX fsm.mc_timer >= fsm.last_mc_timer)`
	Timer Sync FSM Real Clock, with timer limit and no wrap around, LTL based on [6, p. 112], adjusted to address non-determinism	`G ((!has_joined_new_domain & !has_new_backbone_key) -> X fsm.mc_timer >= fsm.last_mc_timer)`

No.	Target Models	CTL/LTL
5.	Timer Sync FSM Integer Clock, with wrap around and `mc_timer_max > 100`, CTL based on [6, p. 112], adjusted to address non-determinism	`AG (((((time_passed >= 100 & received_timer_value <= fsm.mc_timer - latencyTolerance) & !fsm.accepted_frame) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> AX (fsm.sent_msg = TimerNotify_Own_Timer_SerialNr_Tag | fsm.sent_msg = TimerNotify_Own_Timer_Other_SerialNr_Tag))`
	Timer Sync FSM Real Clock, with timer limit, wrap around and `mc_timer_max > 100`, LTL based on [6, p. 112], adjusted to address non-determinism	`G (((((time_passed >= 100 & received_timer_value <= fsm.mc_timer - latencyTolerance) & !fsm.accepted_frame) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> X (fsm.sent_msg = TimerNotify_Own_Timer_SerialNr_Tag | fsm.sent_msg = TimerNotify_Own_Timer_Other_SerialNr_Tag))`
6.	Timer Sync FSM Integer Clock, with wrap around, CTL based on [6, p. 112], adjusted to address non-determinism	`AG ((((((recv_msg != None & received_timer_value <= fsm.mc_timer - latencyTolerance) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) & fsm.notify_timer - time_passed > 0) & AX (((time_passed <= fsm.maxDelayTimeFollowerUpdateNotify & recv_msg = None) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper)) -> AF (fsm.sent_msg = TimerNotify_Own_Timer_SerialNr_Tag | fsm.sent_msg = TimerNotify_Own_Timer_Other_SerialNr_Tag))`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 112], adjusted to address non-determinism	`G ((((((recv_msg != None & received_timer_value <= fsm.mc_timer - latencyTolerance) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) & fsm.notify_timer - time_passed > 0) & X (((time_passed <= fsm.maxDelayTimeFollowerUpdateNotify & recv_msg = None) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper)) -> F[2,2](fsm.sent_msg = TimerNotify_Own_Timer_SerialNr_Tag | fsm.sent_msg = TimerNotify_Own_Timer_Other_SerialNr_Tag))`

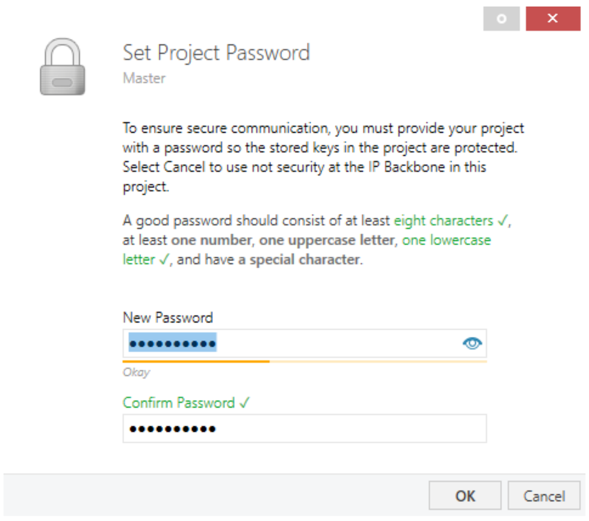
No.	Target Models	CTL/LTL
7.	Timer Sync FSM Integer Clock, with wrap around, CTL based on [6, p. 112], adjusted to address non-determinism	`AG (((((recv_msg != None & received_timer_value <= fsm.mc_timer - latencyTolerance) & fsm.notify_timer - time_passed > 0) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> AX fsm.notify_timer <= fsm.maxDelayTimeFollowerUpdateNotify)`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 112], adjusted to address non-determinism	`G (((((recv_msg != None & received_timer_value <= fsm.mc_timer - latencyTolerance) & fsm.notify_timer - time_passed > 0) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> X fsm.notify_timer <= fsm.maxDelayTimeFollowerUpdateNotify)`
8.	Timer Sync FSM Integer Clock, helper variables and wrap around, CTL based on [6, p. 112], adjusted to address non-determinism	`AG (((((recv_msg != None & received_timer_value <= fsm.mc_timer - fsm.syncLatencyTolerance) & fsm.notify_timer - time_passed > 0) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> AX fsm.is_notify_timer_rescheduled)`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 112], adjusted to address non-determinism	`G (((((recv_msg != None & received_timer_value <= fsm.mc_timer - fsm.syncLatencyTolerance) & fsm.notify_timer - time_passed > 0) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> X fsm.is_notify_timer_rescheduled)`
9.	Timer Sync FSM Integer Clock, with wrap around, CTL based on [6, p. 122], adjusted to address non-determinism	`AG ((((fsm.notify_timer - time_passed <= 0 & recv_msg = None) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> AX (fsm.is_time_keeper & (fsm.sent_msg = TimerNotify_Own_Timer_SerialNr_Tag | fsm.sent_msg = TimerNotify_Own_Timer_Other_SerialNr_Tag)))`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 122], adjusted to address non-determinism	`G ((((fsm.notify_timer - time_passed <= 0 & recv_msg = None) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> X (fsm.is_time_keeper & (fsm.sent_msg = TimerNotify_Own_Timer_SerialNr_Tag | fsm.sent_msg = TimerNotify_Own_Timer_Other_SerialNr_Tag)))`

No.	Target Models	CTL/LTL
10.	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 122], adjusted to address non-determinism	`G ((((((fsm.is_time_keeper & recv_msg != None) & received_timer_value <= fsm.mc_timer - fsm.latencyTolerance) & fsm.notify_timer - time_passed > 0) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) & time_passed > 0) -> X (fsm.is_notify_timer_rescheduled & fsm.notify_timer <= (fsm.maxDelayTimeKeeperPeriodicNotify < fsm.maxDelayTimeKeeperUpdateNotify ? fsm.maxDelayTimeKeeperUpdateNotify : fsm.maxDelayTimeKeeperPeriodicNotify)))`
11.	Timer Sync FSM Integer Clock, with wrap around, CTL based on [6, p. 113], adjusted to address non-determinism	`AG (((((fsm.is_time_keeper & recv_msg = TimerNotify) & received_timer_value > fsm.mc_timer) & fsm.notify_timer - time_passed > 0) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> AX !fsm.is_time_keeper)`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 113], adjusted to address non-determinism	`G (((((fsm.is_time_keeper & recv_msg = TimerNotify) & received_timer_value > fsm.mc_timer) & fsm.notify_timer - time_passed > 0) & !fsm.has_joined_new_domain) & !has_sent_secure_wrapper) -> X !fsm.is_time_keeper)`
12.	Timer Sync FSM Integer Clock, with wrap around, CTL based on [6, p. 134], adjusted to address non-determinism	`AG ((((has_joined_new_domain & has_new_backbone_key) & recv_msg = None) & fsm.notify_timer - time_passed > 0) & !has_sent_secure_wrapper) -> AX (fsm.mc_timer = 0 & fsm.notify_timer = 0))`
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 134], adjusted to address non-determinism	`G ((((has_joined_new_domain & has_new_backbone_key) & recv_msg = None) & fsm.notify_timer - time_passed > 0) & !has_sent_secure_wrapper) -> X (fsm.mc_timer = 0 & fsm.notify_timer = 0))`
13.	Timer Sync FSM Integer Clock, with helper variables and wrap around, CTL based on [6, p. 141], adjusted to address non-determinism	`AG ((((received_timer_value <= fsm.mc_timer - latencyTolerance & !has_joined_new_domain) & recv_msg = None) & fsm.notify_timer - time_passed > 0) & !has_sent_secure_wrapper) -> AX fsm.is_notify_timer_rescheduled)`

No.	Target Models	CTL/LTL
	Timer Sync FSM Real Clock, with timer limit and wrap around, LTL based on [6, p. 141], adjusted to address non-determinism	`G ((((( received_timer_value <= fsm.mc_timer - latencyTolerance & !has_joined_new_domain) & recv_msg = None &) fsm.notify_timer - time_passed > 0) & !has_sent_secure_wrapper) -> X fsm.is_notify_timer_rescheduled)`
14.	Timer Sync FSM Integer Clock	`EF _state_ = SCHED_PERIODIC` `EF _state_ = SCHED_UPDATE`
15.	Timer Sync FSM Integer Clock	`AG !nondeterministic`
16.	Timer Sync FSM Integer Clock	See `timer_sync_fsm_integer_clock.smv` for details

Table 16: Tests for the timer synchronization FSMs

A.3. Device Configuration with the ETS5


Set Project Password

Master

To ensure secure communication, you must provide your project with a password so the stored keys in the project are protected. Select Cancel to use not security at the IP Backbone in this project.

A good password should consist of at least eight characters ✓, at least one number, one uppercase letter, one lowercase letter ✓, and have a special character.

New Password

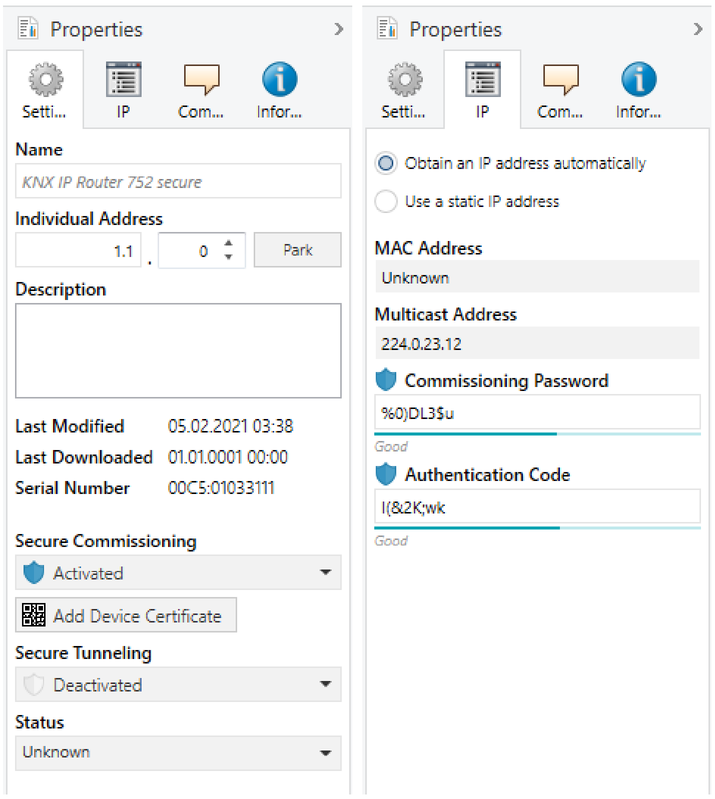
••••••••••

Okay

Confirm Password ✓

••••••••••

Figure 29: Setting a project password
The UI claims it would protect the stored keys



Name
KNX IP Router 752 secure

Individual Address
1.1 . 0

Description

Last Modified 05.02.2021 03:38
Last Downloaded 01.01.0001 00:00
Serial Number 00C5:01033111

Secure Commissioning
Activated

Secure Tunneling
Deactivated

Status
Unknown

( ) Obtain an IP address automatically
( ) Use a static IP address

MAC Address
Unknown

Multicast Address
224.0.23.12

Commissioning Password
%0)DL3$u
Good

Authentication Code
I(&2K;wk
Good

Figure 30: Device properties for KNXnet/IP Secure
“Secure Commissioning” controls use of KNXnet/IP Secure and Data Secure for configuration
“Secure Tunneling” controls use of KNXnet/IP Secure for tunneling
“Commissioning Password” is the password of the management user
“Authentication Code” is the password from which device authentication code is derived

A.4. Test Cases for the ETS5 and KNXnet/IP Secure Routers

ID	Description
`tests.multicast.timerlimit`	A `TIMER_NOTIFY` frame with the maximum mulitcast timer value is sent to the target. The goal is to identify whether a warp-around occurs or if the multicast timer remains stuck at the maximum.
`tests.unicast.toserver.unauthenticatedrequest`	The test software establishes a secure session, but prior to client authentication it attempts to create a KNXnet/IP connection through a `SECURE_WRAPPER` containing a `CONNECT_REQUEST`. The server should refuse with a `SECURE_WRAPPER` containing a `SESSION_STATUS` with the status code `STATUS_UNAUTHENTICATED`.

ID	Description
`tests.unicast.toclient.nowrapper` `tests.unicast.toserver.nowrapper`	For the client, the test software permits the establishment of a secure session and sends a confirmation of the successful authentication with a `SESSION_STATUS` that is not encapsulated in a `SECURE_WRAPPER`. While the client behavior is not well-specified, it should not continue without a correct confirmation of a successful authentication. For the server, the test software establishes a secure session, but attempts to authenticate itself with a `SESSION_AUTHENTICATE` frame that is not encapsulated in a `SECURE_WRAPPER`. The server should reject the frame without a response.
`tests.unicast.toserver.sessionresponse`	A `SESSION_RESPONSE` is sent to the server, which would not occur with a compliant client. The server should reject the frame without a response.
`tests.unicast.toserver.statusauthenticated`	The test software establishes a secure session through a `SESSION_REQUEST` and after receiving a `SESSION_RESPONSE`, it sends a `SECURE_WRAPPER` containing a `SESSION_STATUS` with the status code `STATUS_AUTHENTICATION_SUCCESS` instead of a `SESSION_AUTHENTICATE` frame. The server should reject the frame without a response.
`tests.unicast.toclient.serveractingasclient`	The test software attempts to contact the client like a server, to check if it implements a server as well.
`tests.multicast.wrongmac`	A `TIMER_NOTIFY` frame with incorrect MAC but high timer value is sent. The recipient should reject the frame and not update its multicast timer to match the contained value.
`tests.unicast.toserver.wrongmacauthenticate`	The test software establishes a secure session and attempts to authenticate itself with a `SECURE_WRAPPER` containing a `SESSION_AUTHENTICATE` frame with an invalid MAC. The server should respond with a `SECURE_WRAPPER` containing a `SESSION_STATUS` with the status code `STATUS_AUTHENTICATION_FAILED`.

ID	Description
`tests.unicast.toserver.wrongmacwrapper`	The test software establishes a secure session and attempts to authenticate itself with a `SECURE_WRAPPER`, that has an invalid MAC, containing a `SESSION_AUTHENTICATE` frame. The server should reject the frame without a response.
`tests.unicast.toclient.wrongmac`	The test software permits the client to establish a secure session and authenticate itself. The response is a `SECURE_WRAPPER` with an invalid MAC, containing a `SESSION_STATUS` with the status code `STATUS_AUTHENTICATION_SUCCESS`. The client should reject the frame without a response.
`tests.unicast.toclient.headerwronglength` `tests.unicast.toserver.headerwronglength`	For the client, the test software responds to a `SESSION_REQUEST` with a `SESSION_RESPONSE` that contains an incorrect length in the header. The client should reject the frame. For the server, the test software sends a `SESSION_REQUEST` with an incorrect length in the header. The server should reject the frame without a response.
`tests.unicast.toclient.headerwrongservice` `tests.unicast.toserver.headerwrongservice`	For the client, the test software responds to a `SESSION_REQUEST` with a `SESSION_RESPONSE` that contains an incorrect service type in the header. The client should reject the frame. For the server, the test software sends a `SESSION_REQUEST` with an incorrect service type in the header. The server should reject the frame without a response.
`tests.unicast.toclient.headerwrongversion` `tests.unicast.toserver.headerwrongversion`	For the client, the test software responds to a `SESSION_REQUEST` with a `SESSION_RESPONSE` that contains an incorrect protocol version in the header. The client should reject the frame. For the server, the test software sends a `SESSION_REQUEST` with an incorrect protocol version in the header. The server should reject the frame without a response.

ID	Description
`tests.unicast.toclient.messagefixedsizewronglength` `tests.unicast.toserver.messagefixedsizewronglength`	For the client, the test software responds to a `SESSION_REQUEST` with a `SESSION_RESPONSE` that contains an incorrect length in the header that is significantly larger than the expected value. The client should reject the frame. For the server, the test software sends a `SESSION_REQUEST` with an incorrect length in the header that is significantly larger than the expected value. The server should reject the frame without a response.
`tests.unicast.toclient.sequencenumber` `tests.unicast.toserver.sequencenumber`	For the client, the test software permits to establish a secure session and authenticate itself. Afterwards, a `SECURE_WRAPPER` with an incorrect sequence number, containing a `SESSION_STATUS` with the status code `STATUS_CLOSE`, is sent to the client. The client should reject it and keep the session open. For the server, the test software establishes a secure session and authenticates itself to the server. Afterwards, a `SECURE_WRAPPER` with an incorrect sequence number, containing a `SESSION_STATUS` with the status code `STATUS_CLOSE`, is sent to the server. The server should reject it and keep the session open.
`tests.unicast.toclient.reservedfieldstatus` `tests.unicast.toserver.reservedfieldstatus`	For the client, the test software permits to establish a secure session. Afterwards, a `SECURE_WRAPPER` is sent to the client, containing a `SESSION_STATUS` frame with the status code `STATUS_CLOSE` and a modified reserved field. Since ISO 22510:2019 does not specify validation for `SESSION_STATUS` frame, the client may accept or reject it. For the server, the test software establishes a secure session to the server. Afterwards, a `SECURE_WRAPPER` is sent to the client, containing a `SESSION_STATUS` frame with the status code `STATUS_CLOSE` and a modified reserved field. Since ISO 22510:2019 does not specify validation for `SESSION_STATUS` frame, the server may accept or reject it.

ID	Description
`tests.unicast.toserver.reservedfieldauthenticate`	The test software establishes a secure session and attempts to authenticate itself with a `SECURE_WRAPPER`, containing a `SESSION_AUTHENTICATE` frame with an invalid reserved field. The server should respond with a `SECURE_WRAPPER`, containing a `SESSION_STATUS` with the status code `STATUS_AUTHENTICATION_FAILED`, according to `E02` and `A2` in the session FSM [6, p. 123].
`tests.unicast.toclient.reservedstatus` `tests.unicast.toserver.reservedstatus`	For the client, the test software permits to establish a secure session. Afterwards, a `SECURE_WRAPPER` is sent to the client, containing a `SESSION_STATUS` frame with a reserved status code. The client should reject it. For the server, the test software establishes a secure session to the server. Afterwards, a `SECURE_WRAPPER` is sent to the client, containing a `SESSION_STATUS` frame with a reserved status code. The server should reject it.
`tests.unicast.toserver.hpaiip`	The test software attempts to establish a secure session with a `SESSION_REQUEST` that includes a HPAI, where the port and IP are not set to zero. The server should reject the frame without a response.
`tests.unicast.toserver.hpailength`	The test software attempts to establish a secure session with a `SESSION_REQUEST` that includes a HPAI, where the length field contains an incorrect value. The server should reject the frame without a response.
`tests.unicast.toserver.hpaiprotocol`	The test software attempts to establish a secure session with a `SESSION_REQUEST` that includes a HPAI, where the protocol type is set to UDP. The server should reject the frame without a response.
`tests.unicast.toserver.concurrentsessions`	The test software attempts to establish two secure sessions within the same TCP connection. This should be supported by the server.

ID	Description
`tests.unicast.toserver.reserveduseridauthenticate`	The test software establishes a secure session and attempts to authenticate itself with a `SECURE_WRAPPER`, containing a `SESSION_AUTHENTICATE` frame with a reserved user ID. The server should respond with a `SECURE_WRAPPER`, containing a `SESSION_STATUS` with the status code `STATUS_AUTHENTICATION_FAILED`, according to `E02` and `A2` in the session FSM [6, p. 123].
`tests.unicast.toserver.unuseduseridauthenticate`	The test software establishes a secure session and attempts to authenticate itself with a `SECURE_WRAPPER`, containing a `SESSION_AUTHENTICATE` frame with a user ID that has not been used in the server's configuration. The server should respond with a `SECURE_WRAPPER`, containing a `SESSION_STATUS` with the status code `STATUS_AUTHENTICATION_FAILED`, according to `E02` and `A2` in the session FSM [6, p. 123].
`tests.unicast.toclient.wronglengthwrapper` `tests.unicast.toserver.wronglengthwrapper`	For the client, the test software permits to establish a secure session and authenticate itself. Afterwards, a `SECURE_WRAPPER` with an incorrect length field is sent to the client, containing a `SESSION_STATUS` frame with the status code `STATUS_AUTHENTICATION_SUCCESS`. This is expected to break the processing of the TCP stream and result in a timeout. For the server, the test software establishes a secure session. Afterwards, a `SECURE_WRAPPER` with an incorrect length field is sent to the client, containing a `SESSION_AUTHENTICATE` frame. This is expected to break the processing of the TCP stream and result in a timeout.
`tests.unicast.toserver.sessionrequestudp`	The test software attempts to establish a secure session over UDP. The server should not respond, because UDP is not allowed for KNXnet/IP Secure [6, p. 122, p. 126].
`tests.unicast.toserver.timernotifytcp`	The test software attempts to send a valid `TIMER_NOTIFY` over UDP. The server should reject the frame and not update its multicast timer with the received value, because it was not received through the routing endpoint [6, p. 118].

ID	Description
`tests.unicast.toserver.accesscontrol.management.unwrapped`	The test software establishes a secure session and authenticates itself to the server. Afterwards, a `CONNECT_REQUEST` for a device management connection is sent without being encapsulated in a `SECURE_WRAPPER`. The request should be rejected due to the configured access control settings.
`tests.unicast.toserver.accesscontrol.management.wrapped`	The test software establishes a secure session and authenticates itself to the server. Afterwards, a `SECURE_WRAPPER` is sent, containing a `CONNECT_REQUEST` for a device management connection. The request should only be accepted if the test software authenticates itself as the management user.
`tests.unicast.toserver.accesscontrol.tunnel.unwrapped`	The test software establishes a secure session and authenticates itself to the server. Afterwards, a `CONNECT_REQUEST` for a tunneling connection is sent without being encapsulated in a `SECURE_WRAPPER`. The request should be rejected due to the configured access control settings.
`tests.unicast.toserver.accesscontrol.tunnel.wrapped`	The test software establishes a secure session and authenticates itself to the server. Afterwards, a `SECURE_WRAPPER` is sent, containing a `CONNECT_REQUEST` for a tunneling connection. The request should be accepted, no matter whether the test software authenticates itself as the management user or tunneling user.

Table 17: Test cases for the ETS5 and KNXnet/IP Secure routers

A.5. Test Results for the ETS5

No.	Description
1.	`tests.unicast.toclient.nowrapper` had the unexpected discovery that ETS5 attempts to establish a KNXnet/IP Secure session over UDP, which violates ISO 22510:2019 [6, p. 122 p. 126]. After the reception of the `SESSION_STATUS` that is not encapsulated in a `SECURE_WRAPPER`, the ETS5 still sends a `SECURE_WRAPPER` containing `CONNECT_REQUEST`. While the standard does to specify the validation steps of the client, this does not appear to be correct, as it skips the confirmation of the successful authentication.

No.	Description
2.	`tests.unicast.toclient.serveractingasclient` did not cause a reply by the ETS5, which is correct.
3.	`tests.unicast.toclient.wrongmac` resulted in the ETS5 ignoring the frame with the incorrect MAC, which is correct.
4.	`tests.unicast.toclient.headerwronglength` resulted in the ETS5 ignoring the `SESSION_RESPONSE` with the incorrect length in its header, which is correct.
5.	`tests.unicast.toclient.headerwrongservice` resulted in the ETS5 ignoring the `SESSION_RESPONSE` with the incorrect service type in its header, which is correct.
6.	`tests.unicast.toclient.headerwrongversion` resulted in the ETS5 ignoring the `SESSION_RESPONSE` with the incorrect protocol version in its header, which is correct.
7.	`tests.unicast.toclient.messagefixedsizewronglength` resulted in the ETS5 ignoring the `SESSION_RESPONSE` with the significantly too large length in its header, which is correct.
8.	`tests.unicast.toclient.sequencenumber` resulted in the ETS5 ignoring the `SECURE_WRAPPER` with the old sequence number, which is correct.
9.	`tests.unicast.toclient.reservedfieldstatus` resulted in the ETS5 to accepting the `SESSION_STATUS` despite the incorrect reserved field. This does not violate the specification and is therefore technically correct.
10.	`tests.unicast.toclient.reservedstatus` resulted in the ETS5 ignoring the `SESSION_STATUS`, which is correct.
11.	`tests.unicast.toclient.wronglengthwrapper` resulted in the parsing being broken, which was expected. The ETS5 behaves correctly.

Table 18: Explanation of test results for the ETS5

A.6. Test Results for the KNXnet/IP Secure Routers

No.	Device	Description
1.	SCN-IP100.03	`tests.unicast.toserver.unauthenticatedrequest` resulted in a reply with a `SECURE_WRAPPER` frame, containing a `SESSION_STATUS` with the status code `STATUS_UNAUTHENTICATED`, which is correct.

No.	Device	Description
	KNX IP Router 752 Secure	`tests.unicast.toserver.unauthenticatedrequest` resulted in a reply with a `SECURE_WRAPPER` frame, containing a `SESSION_STATUS` with the status code `STATUS_TIMEOUT`, which is incorrect. It should have triggered event `E05` and perform action `A6`, see session FSM in figure 19.
2.	SCN-IP100.03	`tests.unicast.toserver.nowrapper` resulted in no response, until a timeout occurs, which is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.nowrapper` resulted in no response, until a timeout occurs, which is correct.
3.	SCN-IP100.03	`tests.unicast.toserver.sessionresponse` resulted in the `SESSION_RESPONSE` being ignored, which is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.sessionresponse` resulted in the `SESSION_RESPONSE` being ignored, which is correct.
4.	SCN-IP100.03	`tests.unicast.toserver.statusauthenticated` resulted in no response, until a timeout occurs, which is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.statusauthenticated` resulted in no response, until a timeout occurs, which is correct.
5.	SCN-IP100.03	`tests.unicast.toserver.wrongmacauthenticate` resulted in a reply with a `SECURE_WRAPPER`, containing a `SESSION_STATUS` with the status code `STATUS_AUTHENTICATION_FAILED`, which is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.wrongmacauthenticate` resulted in a reply with a `SECURE_WRAPPER`, containing a `SESSION_STATUS` with the status code `STATUS_AUTHENTICATION_FAILED`, which is correct.
6.	SCN-IP100.03	`tests.unicast.toserver.wrongmacwrapper` resulted in no response, until a timeout occurs, which is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.wrongmacwrapper` resulted in no response, until a timeout occurs, which is correct.
7.	SCN-IP100.03	`tests.unicast.toserver.headerwronglength` resulted in no response and the TCP connection being terminated, which is correct, see [6, p. 41].
	KNX IP Router 752 Secure	`tests.unicast.toserver.headerwronglength` resulted in no response, until a timeout occurs and the TCP connection is eventually closed. This might not be fully correct, since headers that are not well-formed should result in the termination of the TCP connection, see [6, p. 41]. However, it is not specified whether well-formed refers to the structure of the header or also the values of the fields.

No.	Device	Description
8.	SCN-IP100.03	`tests.unicast.toserver.headerwrongservice` resulted in no response, which is correct, until a timeout occurs and the TCP connection is eventually closed. This might not be fully correct, since headers that are not well-formed should result in the termination of the TCP connection, see [6, p. 41]. However, it is not specified whether well-formed refers to the structure of the header or also the values of the fields.
	KNX IP Router 752 Secure	`tests.unicast.toserver.headerwrongservice` resulted in no response, which is correct, until a timeout occurs and the TCP connection is eventually closed. This might not be fully correct, since headers that are not well-formed should result in the termination of the TCP connection, see [6, p. 41]. However, it is not specified whether well-formed refers to the structure of the header or also the values of the fields.
9.	SCN-IP100.03	`tests.unicast.toserver.headerwrongversion` resulted in no response and the TCP connection being terminated, which is correct, see [6, p. 41].
	KNX IP Router 752 Secure	`tests.unicast.toserver.headerwrongversion` resulted in no response, until a timeout occurs and the TCP connection is eventually closed. This might not be fully correct, since headers that are not well-formed should result in the termination of the TCP connection, see [6, p. 41]. However, it is not specified whether well-formed refers to the structure of the header or also the values of the fields.
10.	SCN-IP100.03	`tests.unicast.toserver.messagefixedsizewronglength` resulted in the device being unresponsive to KNXnet/IP Secure frames and required a reboot to restore normal operation. This behavior was caused by a bug in the implementation of firmware version 3.0.3, which permitted effective DoS attacks, hence it is registered as CVE-2021-37740. The bug has been fixed in firmware version 3.0.4.
	KNX IP Router 752 Secure	`tests.unicast.toserver.messagefixedsizewronglength` resulted in no response, until a timeout occurs and the TCP connection is eventually closed. This might not be fully correct, since headers that are not well-formed should result in the termination of the TCP connection, see [6, p. 41]. However, it is not specified whether well-formed refers to the structure of the header or also the values of the fields.
11.	SCN-IP100.03	`tests.unicast.toserver.sequencenumber` resulted in the `SECURE_WRAPPER`, with the incorrect sequence information, being ignored. This behavior is correct.

No.	Device	Description
	KNX IP Router 752 Secure	`tests.unicast.toserver.sequencenumber` resulted in the `SECURE_WRAPPER`, with the incorrect sequence information, being accepted. This behavior is incorrect. However, this does not happen once a sequence number larger than zero has been used.
12.	SCN-IP100.03	`tests.unicast.toserver.reservedfieldstatus` resulted in the `SESSION_STATUS` frame with the incorrect reserved field being accepted. However, since the standard does not specify required validation steps, this behavior is technically correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.reservedfieldstatus` resulted in the `SESSION_STATUS` frame with the incorrect reserved field being accepted. However, since the standard does not specify required validation steps, this behavior is technically correct.
13.	SCN-IP100.03	`tests.unicast.toserver.reservedfieldauthenticate` resulted in the `SESSION_AUTHENTICATE` with the incorrect reserved field being rejected and a `SECURE_WRAPPER` containing a `SESSION_STATUS` with status field `STATUS_TIMEOUT` being sent. This is not correct. According to the session FSM, see figure 19, the event `E02` should have been triggered, because the `SECURE_WRAPPER` is valid, but the `SESSION_AUTHENTICATE` frame is not. Thus, the correct reaction would have been `A2`, sending a `SECURE_WRAPPER` containg a `SESSION_STATUS` with status field `STATUS_AUTHENTICATION_FAILED`.
	KNX IP Router 752 Secure	`tests.unicast.toserver.reservedfieldauthenticate` resulted in the `SESSION_AUTHENTICATE` with the incorrect reserved field being rejected and a `SECURE_WRAPPER` containing a `SESSION_STATUS` with status field `STATUS_TIMEOUT` being sent. This is not correct. According to the session FSM, see figure 19, the event `E02` should have been triggered, because the `SECURE_WRAPPER` is valid, but the `SESSION_AUTHENTICATE` frame is not. Thus, the correct reaction would have been `A2`, sending a `SECURE_WRAPPER` containg a `SESSION_STATUS` with status field `STATUS_AUTHENTICATION_FAILED`.
14.	SCN-IP100.03	`tests.unicast.toserver.reservedstatus` resulted in the incorrect `SESSION_STATUS` frame being ignored, which is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.reservedstatus` resulted in the incorrect `SESSION_STATUS` frame being ignored, which is correct.

No.	Device	Description
15.	SCN-IP100.03	`tests.unicast.toserver.hpaiip` resulted in the `SESSION_REQUEST`, with the incorrect IP address and port in the HPAI, being ignored. This behavior is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.hpaiip` resulted in the `SESSION_REQUEST`, with the incorrect IP address and port in the HPAI, being ignored. This behavior is correct.
16.	SCN-IP100.03	`tests.unicast.toserver.hpailength` resulted in the `SESSION_REQUEST`, with the incorrect length field in the HPAI, being ignored. This behavior is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.hpailength` resulted in the `SESSION_REQUEST`, with the incorrect length field in the HPAI, being ignored. This behavior is correct.
17.	SCN-IP100.03	`tests.unicast.toserver.hpaiprotocol` resulted in the `SESSION_REQUEST`, with the incorrect protocol type in the HPAI, being ignored. This behavior is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.hpaiprotocol` resulted in the `SESSION_REQUEST`, with the incorrect protocol type in the HPAI, being ignored. This behavior is correct.
18.	SCN-IP100.03	`tests.unicast.toserver.concurrentsessions` verified that two KNXnet/IP Secure sessions can be active within the same TCP connection. This behavior is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.concurrentsessions` verified that two KNXnet/IP Secure sessions can be active within the same TCP connection. This behavior is correct.
19.	SCN-IP100.03	`tests.unicast.toserver.reserveduseridauthenticate` resulted in the `SESSION_AUTHENTCATE` frame with a reserved user ID being rejected and a `SECURE_WRAPPER` containing `SESSION_STATUS` with the status field `STATUS_AUTHENTICATION_FAILED` being sent as a reply. This behavior is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.reserveduseridauthenticate` resulted in the `SESSION_AUTHENTCATE` frame with a reserved user ID being rejected and a `SECURE_WRAPPER` containing `SESSION_STATUS` with the status field `STATUS_AUTHENTICATION_FAILED` being sent as a reply. This behavior is correct.
20.	SCN-IP100.03	`tests.unicast.toserver.unuseduseridauthenticate` resulted in the `SESSION_AUTHENTCATE` frame with an unused user ID being rejected and a `SECURE_WRAPPER` containing `SESSION_STATUS` with the status field `STATUS_AUTHENTICATION_FAILED` being sent as a reply. This behavior is correct.

No.	Device	Description
	KNX IP Router 752 Secure	`tests.unicast.toserver.unuseduseridauthenticate` resulted in the `SESSION_AUTHENTCATE` frame with an unused user ID being rejected and a `SECURE_WRAPPER` containing `SESSION_STATUS` with the status field `STATUS_AUTHENTICATION_FAILED` being sent as a reply. This behavior is correct.
21.	SCN-IP100.03	`tests.unicast.toserver.wronglengthwrapper` resulted in a timeout as the TCP stream processing breaks. The behavior is expected and correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.wronglengthwrapper` resulted in a timeout as the TCP stream processing breaks. The behavior is expected and correct.
22.	SCN-IP100.03	`tests.unicast.toserver.sessionrequestudp` resulted in no reply to the `SESSION_REQUEST` sent over UDP, which is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.sessionrequestudp` resulted in no reply to the `SESSION_REQUEST` sent over UDP, which is correct.
23.	SCN-IP100.03	`tests.unicast.toserver.timernotifytcp` resulted in the `TIMER_NOTIFY` frame being ignored, which is correct.
	KNX IP Router 752 Secure	`tests.unicast.toserver.timernotifytcp` resulted in the `TIMER_NOTIFY` frame being ignored, which is correct.
24.	SCN-IP100.03	`tests.unicast.toserver.accesscontrol.management.unwrapped` resulted in the unencapsulated `CONNECT_REQUEST` being reject with an unencapsulated `CONNECT_RESPONSE` with the status code `E_AUTHORISATION_ERROR` as a response. While the rejection is correct, the status code is meant to be used when a client is not authorized to use the IA requested in the extended CRI [6, p. 152]. However, the `CONNECT_REQUEST` did not contain extended CRI, hence this might not be fully correct. ISO 22510:2019 does not appear to specify the intended behavior for this case.

No.	Device	Description
	KNX IP Router 752 Secure	`tests.unicast.toserver.accesscontrol.management.unwrapped` resulted in the unencapsulated `CONNECT_REQUEST` being reject with an unencapsulated `CONNECT_RESPONSE` with the status code `E_NO_MORE_CONNECTIONS` as a response. While the rejection is correct, the status code is meant to be used when the server cannot accept new connections because the maximum number of concurrent connections is already busy [6, p. 152]. However, it could be argued that the maximum number of connections plain KNXnet/IP is reached, which is zero when KNXnet/IP Secure is mandatory. ISO 22510:2019 does not appear to specify the intended behavior for this case.
25.	SCN-IP100.03	`tests.unicast.toserver.accesscontrol.management.wrapped` resulted in a successfully established connection using the management user. For a regular user the `CONNECT_REQUEST` is rejected and a `SECURE_WRAPPER` containing a `CONNECT_RESPONSE` with the status code `E_AUTHORISATION_ERROR` is sent as a response. The overall behavior in both cases is correct, but the status code is meant to be used when a client is not authorized to use the IA requested in the extended CRI [6, p. 152]. However, the `CONNECT_REQUEST` did not contain extended CRI, hence this might not be fully correct. ISO 22510:2019 does not appear to specify the intended behavior for this case.
	KNX IP Router 752 Secure	`tests.unicast.toserver.accesscontrol.management.wrapped` resulted in a successfully established connection using the management user. For a regular user the `CONNECT_REQUEST` is rejected and a `SECURE_WRAPPER` containing a `CONNECT_RESPONSE` with the status code `E_NO_MORE_CONNECTIONS` is sent as a response. The overall behavior in both cases is correct, but status code is meant to be used when the server cannot accept new connections because the maximum number of concurrent connections is already busy [6, p. 152]. This is not the case, because it could accept additional connections through KNXnet/IP Secure, but only for the management user. ISO 22510:2019 does not appear to specify the intended behavior for this case.
26.	SCN-IP100.03	`tests.unicast.toserver.accesscontrol.tunnel.unwrapped` resulted in the unencapsulated `CONNECT_REQUEST` being reject with an unencapsulated `CONNECT_RESPONSE` with the status code `E_CONNECTION_TYPE` as response. This behavior is correct.

No.	Device	Description
	KNX IP Router 752 Secure	`tests.unicast.toserver.accesscontrol.tunnel.unwrapped` resulted in the unencapsulated `CONNECT_REQUEST` being reject with an unencapsulated `CONNECT_RESPONSE` with the status code `E_CONNECTION_TYPE` as response. This behavior is correct.
27.	SCN-IP100.03	`tests.unicast.toserver.accesscontrol.tunnel.wrapped` resulted in a successfully established connection for both the management and regular user. However, there appeared to be a mismatch between the order of user IDs in the ETS5 project files and the user IDs internally assigned by the server.
	KNX IP Router 752 Secure	`tests.unicast.toserver.accesscontrol.tunnel.wrapped` resulted in a successfully established connection for both the management and regular user.

Table 19: Explanation of test results for the KNXnet/IP Secure routers

A.7. Software, Models and Logs

Physical copies of the thesis have a DVD attached that contains the following data:

- NuXMV models
- Source code for the state learner
- Source code for the test software
- Log files for protocol state fuzzing
- Log files for software tests
- Notes about the risk analysis

If you are reading a digital copy and would like access to these files, please send an email to either `guetzkor@informatik.hu-berlin.de` or `rguetzkow@outlook.de`.

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 31. März 2022 ....