Evaluation Method for Smart Home and Smart Grid Authentication Protocols

Master's thesis

Student: Marvin Uku
182509IVCM
Supervisor: Olaf Manuel Maennel, Ph.D
E-mail: marvin.uku@ttu.ee
Study programme: Cyber Security

Tallinn 2020

Table of Contents

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis and this thesis has not been presented for examination or submitted for defence anywhere else. All used materials, references to the literature and work of others have been cited.
Author: Marvin Uku
05.11.2020

Abstract

Authentication is an essential part of accessing network resources and therefore necessary in every network. It is the verification of identity which is attached to some kind of an identifier, which is presented to an authority or other participant in the communication to access needed service or resource. In this work smart grid and smart home authentication protocols evaluation method is proposed. Evaluation is done according to 5 categories - security, efficiency, audit, administration and transport. These categories contain 27 criterion in total, three of them contain formal verification of a security protocol using BAN logic and ProVerif cryptographic protocol verifier. All of the mentioned categories have different weights in the evaluation method, each of the criterion results and boolean result, TRUE or FALSE. Final score is calculated according to these criterion total sum divided with the total sum of the weights, giving the result from 0 to 1, one being the best and zero worst choice for smart home and smart grid network.
6LOWPAN is found to be most secure according to this method, following WPA3. In this work depreciation of X10 and IrDA is proposed according to the results gained from this method.
The thesis is in English and contains 323 pages of text, 8 chapters, 17 figures, 12 tables.

Annotatsioon

Autentimine on üks põhilistest osadest võrguressursi ligipääsu protsessi juures ja seega oluline osa igas võrgus. See on identiteedi verifitseermine mingi kindla tunnuse abil, mille ettenäitamist on vaja, et pääseda ligi mingile teenusele või ressursile võrgus. Selles töös pakutakse välja hindamismeetod nutivõrgu ja targakodu autentimisprotokollidele. Selle meetodi abil hinnatakse protokolli viie kategooria abil - turvalisus, tõhusus, auditeermine, administreerimine ja transport. Need kategooriad jagunevad omakorda 27-ks erinevaks kriteeriumiks, millest kolm on turvaprotokolli ametlik verifitseerimine kasutades BAN loogikat ja ProVerifi krüptograafilist protokolli kinnitajat. Kõik eelnevalt nimetatud kategooriad omavad erinevaid kaalusid. Iga kriteerium on hinnatud tõeväärtusega, kas õige või vale. Lõpptulemus saadakse, kui nende kriteeriumite summa jagada kaalude kogu summaga, mis annab väärtuseks hinde nullist üheni, milles üks on parim ja null kõige halvem valik nutivõrgu ja targakodu võrgus autentimiseks.
Selle meetodi tulemusena on kõige turvalisem 6LOWPAN protokoll, sellele järgneb WPA3-l baseeruvad ühendused. Selles töös soovitatakse samuti kaaluda X10 kui ka IrDA protokollide kasutamist. Kui nende kasutamist jätkata, siis need protokollid vajavad suurt arendamist. Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 323 leheküljel, 8 peatükki, 17 Figuret, 12 tabelit.

List of Abbreviations and Terms

BAN logic - Burrows–Abadi–Needham logic
MAC - Medium Access Control
IP - Internet Protocol Address
IMSI - International Mobile Subscriber Identity
SIM - Subscriber Identification Module
PIN - Personal Identification Number
NAS - Network Access Server
PAP - Password Authentication Protocol
CHAP - Challenge Handshake Authentication Protocol
EAP - Extensible Authentication Protocol
IoT - Internet of Things
ISP - Internet Service Provider
Kerberos - Network authentication protocol
IR - Infrared
IrDA - Infrared Data Association
IrDA SIR - IrDA Serial Infrared
IrPHY - IR Physical Layer
IrLAP - IrDA Link Access Protocol
IrLMP - IrDA Link Management Protocol
LM-MUX - Link Management Multiplexer
LSAP - Link Service Access Point
RF - Radio Frequency
TDMA - Time-Division Multiple Access
FDMA - Frequency-Division Multiple Access
CDMA - Code-Division Multiple Access
OFDMA - Orthogonal Frequency-Division Multiple Access
AKA - Authentication and Key Agreement
ETSI - European Telecommunications Standards Institute
GSM - Global System for Mobile Communication
SMS - Short Messaging Service
HSCSD - High-Speed Circuit-Switched Data
GPRS - General Packet Radio Service
SGSN - Service GPRS support node

GGSN - Gateway GPRS support node
EDGE - Enhanced Data Rates for GSM Evolution
CDPD - Cellular Digital Packet Data
UMTS - Universal Mobile Telecommunications System
3GPP - 3rd Generation Partnership Project
FDD - Frequency Division Duplex
TDD - Time Division Duplex
UTRA - UMTS Terrestrial Radio Access
W-CDMA - Wideband-Code Division Multiple Access
EPS-APA - Evolved Packet System Authentication and Key Agreement
ISIM - IP Multimedia Services Identity Module
LTE - Long Term Eveolution
RFID - Radio-Frequency Identification
CCM - Cipher Block Chaining
AES - Advanced Encryption Standard
CBC-MAC - Cipher Block Chaining Message Authentication Code
CTR - Counter
WPAN - Wireless Personal Area Network
AD hoc network - A network that is composed of individual devices communicating with each other directly
Piconet - An ad hoc network that links a wireless user group of devices using Bluetooth technology protocols
Scatternet - A type of ad hoc computer network consisting of two or more piconets
FSK - Frequency-shift keying
EHW - Energy Harvested Wireless protocol
OSI - Open Systems Interconnection model
RLC - Rolling Code
SLF - Security Level Format
CMAC - Cipher-based Message Authentication Code
AMI - Advanced Metering Infrastructure
NIST - National Institute for Standards and Technology
SEP - Smart Energy Profile
PHY - Physical Layer
CSMA-CA - Carrier-Sense Multiple Access with Collision Avoidance
ACK - Acknowledgement

ECDH - Elliptic Curve Diffie-Hellman

CRC - Cyclic Redundancy Check

PSK - Pre-Shared Key

MIMO - Multiple Input, Multiple Output

WEP - Wired Equivalent Privacy

WPA - WiFi Protected Access

TKIP - Temporal Key Integrity Protocol

CCMP - Counter Cipher Mode with Block Chaining Message Authentication Code Protocol

TLS - Transport Layer Security

TTLS - Tunneled Transport Layer Security

PAP - Password Authentication Protocol

PEAP - Protected Extensible Authentication Protocol

AP - Access Point

STA - Station

AKM - Authentication and Key Management

FQDN - Fully Qualified Domain Name

UWB - Ultra-Wide Band

BSK - Binary Shift Keying

WiMAX - Worldwide Interoperability for Microwave Access

ISO - International Organization for Standardization

SS - Subscriber Station

BS - Base Station

SA - Security Association

TEK - Temporary/Transmission Encryption Key

HMAC - Hash function-based Message Authentication Code

DES - Data Encryption Standard

RSA - Rivest, Shamir, Adleman algorithm

PKM - Privacy Key Management

SDSL - Symmetric Digital Subscriber Lines

ADSL - Aymmetric Digital Subscriber Lines

LLC - Logical Link Control

PLC - Powerline Communication

UPB - Universal Powerline Bus

PPP - Point-to-Point Protocol

CENELEC - European Committee of Electrotechnical Standardisation
FHSS - Frequency Hopping Spread Spectrum
CCA - Clear Channel Assessment
MAU - Multiattribute Utility
VAES - Variable AES
KEK - Key Encryption Key
PAK - Primary Authorization Key
DTLS - Datagram Transport Layer Security
LAN - Local Area Network
WLAN - Wireless Local Area Network
DSK - Device-Specific Key
J-PAKE - Password Authenticated Key Exchange by Juggling
CoAP - Constrained Application Protocol
SSL - Secure Socket Layer
CA - Certificate Authority
MS - Microsoft
PPTP - Point-to-Point Tunneling Protocol
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
UEA - UMTS Encryption Algorithm
UIA - UMTS Integrity Algorithm
EIA - EPS Integrity Algorithms
EEA - EPS Encryption Algorithms
PTK - Pair-wise Temporal Key
GTK - Group Temporal Key
MKID - Master Key Identifier
MIC - Message Integrity Code

List of Figures

List of Tables

1 Introduction

Authentication is the verification of the identity or a subject performing an action. The identity can be personal, logical, like a user ID and password, bound to an infrastructure like an IP-Address, or bound to a device, like a Medium Access Control (MAC) address or the International Mobile Subscriber Identity stored in the SIM (Subscriber Identification Module) Card. The subject of authentication can be a service user or a service provider. [13]
Authentication mechanisms can be classified as follows: [13]

1. Knowledge-based authentication founds on the knowledge of shared secrets, such as PINs (Personal Identification Number) and passwords.

2. Cryptography-based authentication includes digital signatures, challenge-response mechanisms, and message authentication codes. The user owns a private key as a characteristics.

3. Authentication based on biometrics uses inherent informations on subjects like fingerprint, voice, and eye characteristic.

4. Authentication based on secure tokens binds the subject to some kind of ownership, e.g. the ownership of a smart card. It is combined mostly with cryptographic mechanisms to transfer the information on the token to the authenticator.

5. Digitized signatures, including digital images of handwritten signatures and signature dynamics (i.e., measurements of the direction, pressure, speed, and other attributes of a handwritten signature) are not widely used so far.

Authentication protocols are used in establishing a data-link layer connection, mostly a dial-up connection between an end-user's host and the Network Access Server (NAS), but also for switched lines. In general, they allow a peer to transmit authentication information to the authenticator until the authenticator acknowledges the peer. In PAP (Password Authentication Protocol) the authentication is based on a pair of user name and password. CHAP (Challenge Handshake Authentication Protocol) [14] supports a challenge response mechanism, which is controlled by the authenticator. In a challenge response mechanism the password does not have to be transmitted over the link. EAP (Extensible Authentication Protocol) [15] supports authentication based on different

mechanisms, identity and challenge-based, but also using One Time Passwords or Generic Token Cards. These protocols are often integrated in the protocols at the transport level, which implement authentication-based authorization. [13]
The goal of this study is to develop a method to evaluate authentication protocols for smart home and smart grid networks, with the scope of all prominent protocols. The outcome should be applicable method to evaluate authentication methods for existing and in-development smart home and smart grid protocols. As mentioned earlier this method could lead the developers to the path of unified protocol, which would assist in communicating with different service providers and facilities nearby, when needed in the future.
This work will show different shortcomings and advantages of these protocols and methods of authentication, as well as help smart home and smart grid developers to choose which protocol to implement for authentication and communication for smart home and smart grid networks. This research will also point out some main threat vectors for authentication in these networks from the security specialist's point of view. It might point out some key aspects in this field for developers, which needs to be taken into consideration. Main contribution of this research is working applicable evaluation method for protocol authentication methods. Another contribution is the evaluation of the protocols used in this work.

2 Problem Statement

Internet of Things devices are being developed so fast, that for example smartphones are more powerful than computers first used to authenticate themselves in network for resource access. IoT device authentication is still relevant security topic. The growth of using IoT devices is expected to reach 100 billion in 2020. [16] With the rapid increase of devices and users, authentication becomes more relevant than ever before. Each device and user need unique identity which can be verified, for devices IP or MAC address, for users username and password, token or something that would verify their identity. With these in place, administrators and service provides, can ensure secure communication as well as prevent devices from harmful actions. Authentication protocols are the foundation of security in distributed systems, and therefore it is essential that these protocols function correctly [17].
Node authentication is necessary to prevent illegal access to system. Authentication mechanisms provide integrity and confidentiality as well, there for authentication is necessary and must-have in used system. It is as essential as having a key to open a door to one's home. Without authentication, door is without a key and anyone could access it.
Smart home is a building automation, which involves the control and automation of all its embedded technology. Smart home has appliances, lighting, heating, air conditioning, TVs, computers, entertainment systems, big home appliances such as washers/dryers and refrigerators/freezers, security and camera systems capable of communicating with each other and being controlled remotely by smartphone, computer or really any Internet of Things device. [18] According to the research [19] done by Juniper Research, there will be 1.3 billion automation and/or monitoring smart homes by 2024.
Smart grid is an electrical grid whose operation has been transformed from a twentieth century analog technology base to the pervasive use of Digital Technology for communications, monitoring (e.g., sensing), computation, and control. [20] Smart grid can be also controlled remotely. With the development of smart grid networks, there are more industrial uses for these connections and even for smart cities soon. It is expected to have 90 million units of smart grid meter installations only in the Unites States of America by the end of this year [21].
Regarding to the use of these devices and networks, the need for security grows as well. In the field of smart home and grid, network devices accessing network resources [22] are not authenticated for each access of needed network resource at all or it is done in the way, which is not considered secure nowadays [23]. For example not using relevant

cryptographical standards or protocols which are being used, do not support these standards. In the past when smart homes were first developed, IoT devices were not powerful enough to implement relevant cryptography. Now this should be in the past and now IoT device's processors have enough computing power to encrypt and decrypt fast enough to hold encrypted communication and key exchanges during communication. If there is no method to evaluate smart home and smart grid protocols, there cannot be unified smart home or smart grid protocol for different Internet Service Providers and electricity companies to use for interoperable communication. To evaluate protocols and deciding the best and secure protocol to use in the network, there has to be deep and detailed analysis of existing protocols or standard, according to which protocol will be developed. There are comparisons between different smart home and smart grid protocols, but not detailed enough. These comparisons have not been fully analysed. For example [24], [25], [20], [26], [27], [28], [29] which have compared different protocols' metrics, but not full examination.
Deeper analysis and comparison of smart home and smart grid protocols is needed. Every category which is compared, comes with pros and cons, not just in plain text. There are reasons for these implementations and these reasons should be analysed. Some tradeoffs can be eliminated with other comparison and evaluation categories. For example from which network layer protocol uses to send data or access resource or even according to which standard cryptography is implemented and how it is done.
From comparison evaluation method should be developed to understand which protocols to prefer in smart home and smart grid environments. This method considers all needed features and characteristics of applied protocols. It should be applicable to any smart home and smart grid network protocol. This evaluation method could point developers in the way of unified protocol. This protocol could be used by Internet Service Provider and electricity company in the region, so that nearby facilities and establishments could function with same manufacturer's devices and service provider.

3 Background

3.1 Smart Home

A smart home is a convenient home setup where appliances and devices can be automatically controlled remotely from any internet-connected IoT device. Smart homes offer a more convenient and better quality of life by introducing automated appliance control and helpful services. Smart homes optimize user comfort by using context awareness and predefined constraints based on the conditions of the home environment. A user can control home appliances and devices remotely, which enables him or her to execute tasks before arriving home.For example, most common appliances in smart home are Smart TV, wireless speakers, lighting devices, doors, cameras, thermostats and even refrigerators. Earliest smart home devices date back as far as 1998. Ambient intelligence systems, which monitor smart homes, sometimes optimize the household's electricity usage. Smart homes enhance traditional security and safety mechanisms by using intelligent monitoring and access control. [30] From smart home technology, the possibility of smart cities can grow, which means that the network will grow exponentially, where the devices are connected. When all this might be more convenient, but this means that security problems is much bigger problem in this field than ever before.

3.2 Smart Grid

To tackle the challenges of the existing power grid, the new concept of smart grid has risen. The smart grid can be considered as a modern electric power grid infrastructure for enhanced efficiency and reliability through automated control, high-power converters, modern communications infrastructure, sensing and metering technologies, and modern energy management techniques based on the optimization of demand, energy and network availability. [31] The new smart grid needs much more complex and different infrastructure than current one. A communications system is the key component of the smart grid infrastructure. It is critical for electric utilities to define the communications requirements and find the best infrastructure to handle the output data and deliver a reliable, secure and cost-effective service throughout the total system. Different communications technologies, wired, wireless and both, can be used for data transmission

between smart meters and electric utilities. In some instances, wireless communications have some advantages over wired technologies, such as low-cost infrastructure and ease of connection to difficult or unreachable areas. On the other hand, wired connections have no interference problems. [32]
The challenge is that the smart grid systems are lacking globally accepted standards and this situation prevents the integration of advanced applications, smart meters, smart devices, and renewable energy sources and limits the interoperability between them [33]. Both smart home and smart grid networks are are connected to internet and allow remote control of the connected devices, with the difference that a smart grid sends electricity as well with the same connection.

4 Literature Review

There are a lot of metrics comparisons between different protocols used by smart home and smart grid, for example [28], [26], [25], [24], [34], [27], [20], but there are no comparisons between all the authentication protocols used for smart grid and smart homes. Also, no unified authentication method in this field. Authentication is mostly done once, when connected to the network, and never again. At least most of the protocols which are widely used, support encryption, but some are lacking in this field and should not even be used for sensitive data in open networks.
For example Kerberos can be used for authentication in smart home and smart grid networks. [35]. This raised questions for the author, that why has not been used Kerberos or any other more mature authentication method for smart homes or smart grid networks before. Now when Kerberos authentication is feasible for IoT devices to use, author will take this as a threshold to evaluate authentication. It is even more relevant for smart grid networks, because Kerberos is used widely in organizations' and companies' networks and smart grid networks are to manage by end users, so the difficulty of Kerberos configuration should not be an issue. Kerberos has been suggested for authentication with biometric templates [36]. This approach provides another aspect to authentication for network security, because biological attributes are difficult to reproduce [36]. In resource constrained networks, for example smart grid network, constrained application protocol has been proposed using Kerberos [29]. This proposed authentication protocol adds less bits to original message than other used encryption schemes [29]. Due this, it reduces packet size decreases the communication overhead, also these parameters show that security performance is improved as well [29].
More popular and used smart home and smart grid technologies' authentication protocols have been selected for this work. Proposed method is applicable to any smart home and smart grid network authentication protocol and the result will be equal to these, which were evaluated in this work.

4.1 Literature Review Method

First goal was to find out which smart home solutions are there. For this following search terms were used: "smart home solutions", "smart home platforms". Second goal was to find out which protocols were used for these smart home solutions. The

search term used for this was “smart home protocols”. Next search terms were “wireless smart home protocols” and “wired smart home protocols”. After receiving results for these questions search for these protocols began. Key terms for these searches were “name communication protocol”, “name protocol”, “name protocol for smart home”, “name protocol in smart home networks”, “name protocol IOT”, “name protocol design”, “name protocol standard”, “name protocol comparison”. Third goal was to find out authentication methods of these smart home solutions. Following search terms were used “name protocol authentication”, “name system authentication”, “name solution authentication”, “name device authentication”, “name protocol authentication in smart home network”, “name protocol authentication with IoT devices”.
More searched terms are available in Appendix 28.
Main research gap while conducting the review was identified, which is that there is no unified authentication method or protocol standard for smart homes or smart grid networks. Nowadays IoT devices are being developed so fast, that these devices are catching up with computer computing power. In the past when smart homes were first developed, IoT devices were not powerful enough to implement relevant cryptography. Now this should be in the past and phone processors have enough computing power to encrypt and decrypt data fast enough to have encrypted communication.

4.2 Wireless

Wireless connection is used by a computer network which uses radio frequencies for communication. There are different area networks, for example personal, local, wide, metropolitan. There is also mesh, cellular, global area and space. Personal area network connects devices in a personal workspace, local area network interconnects devices in a limited area, for example inside a school, wide area network is primally used by businesses, education and government entities, metropolitan area network is a network, which connects users and devices in a geographic region, mesh network is a local network that connects network nodes directly, dynamically and not hierarchically to each other. Cellular network connects voice and data devices with capable connectivity to public telephone network and Internet. Most of the inhabited area of Earth has been covered with needed radio towers by service providers, global area network is composed of different networks. There are different technologies for each of these networks, these connections are introduced in next sections.

4.2.1 Infrared

The infrared (IR) is a fast wireless information transfer. IR optical medium has been restricted to the room of operation, of being spectrally unregulated and of providing potentially very high data rates, but optical power output is limited by eye-safety regulations and a desire to limit power consumption [37]. Wireless IR is particularly suited for short-range indoor applications [38]. The Infrared Data Association was created in 1993 to establish an open standard for short-range IR data communication, IrDA SIR (serial infrared) protocol standard was developed. It provided a simple, low cost and reliable means of data communication between IrDA compliant devices using point-to-point half-duplex IR links. [37].

4.2.1.1 IrDA protocol

IrDA protocol stack consists of three mandatory layers: the physical (IrPHY) layer, the IrLAP layer, and the IrDA Link Management Protocol (IrLMP) layer. [37]
The IrPHY layer specifies the physical hardware for the IR link including IR transmitter and receiver, filters, and modulation and encoding hardware. [37]
The IrLAP layer is an HDLC-based data link layer providing device discovery, link establishment and shutdown, and reliable data exchange. [37]
The IrLMP layer consists of two distinct elements. The link management multiplexer (LM-MUX) provides a means for multiple entities on a device to independently and simultaneously a single established IrLAP link. The layer interacts with higher levels of the protocol using link service access points (LSAPs). [37]
IrDA functions in the range of 1-3 meters and operates over 850nm – 900 nm frequency. Its maximum data rate is 100 MB/s and it needs a line of sight to function. [39] IrDA does not provide any link-level security, so there is no authentication or authorization and all information is sent unencrypted. If authentication/authorization/encryption is needed, it has to be implemented at software level. IrDA supports only Point-to-Point connections and requires direct line-of-sight between two IrDA devices. [3] IrDA does not provide any link-level security, so there is no authentication or authorization, and all information is sent unencrypted. If authentication/authorization/encryption is needed, it has to be implemented at software level. IrDA supports only Point-to-Point connections,

and requires direct line-of-sight between two IrDA devices. So, in spite of lacking support for explicit security measures, IrDA can be considered as a relatively secure technology. On the other hand, IrDA lacks the convenience of wireless RF technologies such as Bluetooth and WLAN.

4.2.2 Cellular

Cellular or mobile network is wireless communications network distributed over a limited land area that includes at least one fixed position transceiver. Cellular network uses time-division multiple access (TDMA), frequency-division multiple access (FDMA), code-division multiple access (CDMA), and orthogonal frequency-division multiple access (OFDMA) medium access controls for communication. Cellular Networks have been around since the 1980s. First generation (1G) networks were the first cellular networks introduced in the 1980s. They were only capable of transmitting voice at speeds of about 9.6 kbps max, because of that limitation, 1G networks are not reviewed in this work. [40] Cellular connections use AKA (Authentication and Key Agreement) for authentication [41]:

4.2.2.1 GSM

GSM stands for Global System for Mobile Communication. It is a digital cellular technology used for transmitting mobile voice and data services. GSM is a standard, which is developed for second generation mobile networks (2G). It was developed by European Telecommunications Standards Institute (ETSI). 2G uses different data rates in all countries, depending of the service provider. It can achieve data rate of 270 kbps. [40] EAP-SIM is an authentication protocol, which has been developed for GSM for authenticated key access. [42]
GSM involves services :

1. Short Messaging Service (SMS): Transfer of messages between cell phones. Large messages are truncated and sent as multiple messages.
2. High-Speed Circuit-Switched Data (HSCSD): This was the first attempt at providing data at high speeds data over GSM, with speeds of up to 115 kbps. This technique cannot support large bursts of data. GPRS was adopted instead of HSCSD.

3. General Packet Radio Service (GPRS): This technique can support large bursty data transfers. In order to support this two new elements have to be added to existing networks. Service GPRS support node (SGSN) for security mobility and access control and Gateway GPRS support node (GGSN) in order to connect to external packet switched networks.

4. Enhanced Data Rates for GSM Evolution (EDGE): The standard GSM uses GMSK modulation. Edge uses 8-PSK modulation. GPRS and EDGE combined provide data rates of up to 384 kbps.

5. Cellular Digital Packet Data (CDPD): CDPD is a packet based data service. CDPD is able to detect idle voice channels and uses them to transfer data traffic without affecting voice communications.

GSM authentication is described in Appendix 14.

4.2.2.2 UMTS

[43] Universal Mobile Telecommunications System (UMTS) is a standard for third generation digital cellular networks. It is based on GSM. UMTS was developed by 3GPP (3rd Generation Partnership Project) and uses different data rates in all countries, depending of the service provider. UMTS can support maximum data transfer rate of 42 MB/s. The radio access specifications provide for Frequency Division Duplex (FDD) and Time Division Duplex (TDD) variants, and several chip rates are provided for in the TDD option, allowing UTRA technology to operate in a wide range of bands and co-exist with other radio access technologies. UMTS includes the original W-CDMA scheme using paired or unpaired 5 MHz wide channels in globally agreed bandwidth around 2 GHz, though subsequently, further bandwidth has been allocated by the ITU on a regional basis.
UMTS uses EPS-AKA [43]:

1. A shared secret K has been established earlier between the ISIM and the Authentication Centre (AuC). The secret is stored in the ISIM.

2. The AuC of the home network generates an authentication vector AV, based on the shared secret K and a sequence number SQN. The authentication vector contains a random challenge RAND, authentication token AUTN, expected authentication

result XRES, a session key IK for integrity check, and a session key CK for encryption.

3. The authentication vector is sent to a server. Server can also have more than one AVs.

4. The server creates an authentication request, which contains the random challenge RAND, and the authentication token AUTN.

5. The authentication request is sent to the client.

6. Using the shared secret K and the sequence number SQN, the client verifies the AUTN with the ISIM. If the verification is successful, the network has been authenticated. The client then produces an authentication response RES, using the shared secret K and the random challenge RAND.

7. The authentication response, RES, is delivered to the server.

8. Server compares the authentication response RES with the expected response XRES. If the two matches, the user has been successfully authenticated, and the session keys, IK and CK, can be used for protecting further communications between the client and the server.

UMTS authentication is described in Appendix 15.

4.2.2.3 LTE

[43] LTE (Long Term Evolution) is a standard for wireless broadband communication for mobile devices. This standard is based on previous UMTS and GSM standards. 4G uses different data rates in all countries, depending of the service provider. LTE data rate can peak over 200 MB/s. LTE is different from other technologies that call themselves 4G, because it is completely integrated into the existing cellular infrastructure. his allows seamless handoff and complete connectivity between previous standards and LTE. [44] The standard is designed for full-duplex operation, with simultaneous transmission and reception. [44]
LTE uses same same authentication protocol as UMTS, which is described in Appendix 15. [10]

4.2.3 Radio Frequency

4.2.3.1 Dash7

Dash7 (Developers' Alliance for Standards Harmonization of ISO 18000-7) The protocol is intended for RFID (Radio-Frequency Identification) and wireless sensor networks. Dash7 operates in the 433 MHz frequency band, achieves a data rate of 27.8 kbps and reaches up to 250 m. [39] [45]
There are four different device classes defined in D7A (Dash7 Alliance Protocol). [46]

1. Blinker - Transmits and does not use a receiver.
2. EndPoint - Transmits and receives data.
3. Subcontroller - Full featured device, not always active.
4. Gateway - It connects D7A network with the other networks. It is always online. It always listens, unless it is transmitting.

All devices in the Dash7 network support one or more of the these device classes. Dash7 supports two communication models: pull and push. The dialogues between tags and interrogators are query-response based (pull model). This request-response mechanism is described by the D7A query protocol. Data transfer initiated from the tags to the gateway, on the other hand, is based on the push model. This approach is implemented as an automated message or a beacon which is sent on specific time intervals. This system is called Beacon Transmit Series. DASH7 defines two types of frames: a foreground frame and a background frame. The foreground frames are regular messages which contain data or its requests. Background frames on the are very short broadcast messages. Background frames are used by the D7A advertising protocol for rapid ad-hoc group synchronization. . [47], [48]
Dash7 uses CCM for authenticate-and-encrypt block cipher mode. By using 128-bit block cipher AES, CCM can be operated. To compute the authentication field, the CBC-MAC mode is used and to encrypt the message data, the counter (CTR) mode is used. [48], [49].
CCM (CBC-MAC) authentication has been shown in Appendix 20.

4.2.3.2 Bluetooth

Bluetooth is standardized according to IEEE 802.15.1, which is based on a wireless radio system designed for short-range. Bluetooth functions as wireless personal area network. Two connectivity topologies are defined in Bluetooth: the piconet and scatternet. A piconet is a Wireless Personal Area Network (WPAN) formed by a Bluetooth device serving as a master in the piconet and one or more Bluetooth devices serving as slaves. A frequency-hopping channel based on the address of the master defines each piconet. A scatternet is a collection of operational Bluetooth piconets overlapping in time and space. Two piconets can be connected to form a scatternet. [50] Bluetooth uses 2.4 GHz frequency band, and has a maximum signal rate of 1 MB/s with the nominal range of 10 meters. It also uses FSK modulation and supports E0 stream cipher, shared secret authentication and 16-bit CRC data protection [50].
In Bluetooth Generic Access Profile, the Bluetooth security is divided into three security modes: [51]

1. Non-secure.
2. Service level enforced security.
3. Link level enforced security.

The difference between security mode 2 and security mode 3 is that in security mode 3, the Bluetooth device initiates security procedures before the channel is established. There are also different security levels for devices and services. For devices, there are 2 levels, "trusted device" and "untrusted device". The trusted device obviously has unrestricted access to all services. For services, 3 security levels are defined: services that require authorization and authentication, services that require authentication only and services that are open to all devices. [51]
Bluetooth authentication is described more precisely in Appendix 1.

4.2.3.3 EnOcean Serial Protocol 3

Originally developed by Siemens AG but became an open protocol at 2008 when EnOcean Alliance was formed by EnOcean. EnOcean Serial Protocol 3 (ESP3) is based on global standard based on International Electrotechnical Commission (IEC) standard ISO/IEC 14543-3-10 for low-energy wireless applications. ESP3 uses point-to-point communication

with the maximum range of 30 meters. ESP messages are protected by AES-128 encryption [52], only mutual authentication means are available. The EHW security is implemented on the OSI presentation layer of the EHW protocol stack. A message contains all fields that a telegram may have: the R-ORG, DATA, Sender ID, receiver ID, repeater counter as well as the security specific members like RLC, CMAC, SLF. [4]
EnOcean authentication is described in Appendix 2.

4.2.3.4 ANT+

ANT is a proprietary wireless sensor network protocol. ANT operates on a 2.4GHz frequency and provides a maximum data rate of 1Mbps. [39] Its primary current use is for device-to-device communication between Master devices, typically sensors such as heart rate monitors or geocaching chips, and slaves, such as ANT-enabled watches and cell phones which process sensor data. However, the protocol supports more than a one-to-one master-slave relationship; it supports star, tree, and mesh topologies. [53]
ANT typically operates in burst mode, with 64-bit packets of information. Each packet contains header information necessary for message transmission and a check sum to verify message contents. Optionally, ANT can operate in an authenticated mode which allows for the acknowledgement of messages. However, this method simply adds an ACK reply, based on the check sum, from receiving devices; it does not add a cryptographically-secure MACs. Additionally, ANT offers an advanced burst mode of 128-bit packet size that draws more power. A 64-bit Network Key is required to initiate a channel. This key only secures the creation of the channel; it does not encrypt messages sent within the channel. ANT supports an 8-byte network key and 128-bit AES encryption for ANT master and slave channels, but is not required by default. If further security is required, authentication and encryption can be implemented through the application level. [53]
ANT+, by Dynastream Innovations, builds on the ANT protocol by standardizing device profiles, which are set parameters for a list of devices, such as a stride based speed and distance monitor. These profiles assign each type of device to a specific frequency within the ANT band as well setting more technical details, such as the other requirements for initiating a channel. Furthermore, in order to allow for easy interoperability between the various sensors and possible slave devices (phones, watches, etc). ANT+ dictates a centrally-managed scheme for Network Keys, allowing devices to easily connect to one another at the cost of the security benefits provided by the Network Key. [53]
ANT+ is as well unencrypted by default. The protocol appears to offer confidentiality through two mechanisms: RF frequency/Channel ID and a network key. However, this

system only prevents legitimate users from receiving data from unintended IoT devices. It does nothing to prevent malicious actions, as the ANT+ frequencies are assigned by master devices profile. Each ANT+ packet can be encrypted with a 64-bit Network Key. However, due to the relative short length of this key, and the deterministic nature of the encryption function, this system does not provide adequate security. threats to confidentiality could be prevented through use of ANT+s optional encryption, AES-128 in CTR mode. Unfortunately, there are three usage cases that severely impede the usage of ANT+s AES encryption [53]:

Multi-node networks ANT+ prefers to use multichannel communication to support multi-node network topologies. However, AES encryption cannot be used in multichannel mode, forcing the usage of single channel communications. While single channel schemes do support multi-node topologies, it is not power inefficient any more, as all Master devices must operate in continuous scanning mode, which draws significant power and should not be used for devices that have tight power constraints. [53]

Low power applications ANT+ requires the advanced burst method of communication with AES encryption, which uses again more power than the traditional burst mode. The AES computation itself is power intensive relative to other algorithms [53].

Low cost or legacy applications In low cost applications, it may not be feasible to implement AES. Moreover, AES capability is a recent development, so older ANT+ processors also lack the capability, forcing implementations that require backwards compatibility to forego AES encryption. [53]

ANT+ provides no cryptographic authentication. However, experimental setup has been done in [54] and CMAC was found to be most efficient for the purpose of ANT+.

4.2.3.5 ZigBee SEP Protocol

ZigBee is wireless communication protocol, which is based standardized protocol of IEEE 802.15.4, which is for low-rate wireless personal area networks. It is low in power usage, data rate, complexity and cost of deployment [31]. ZigBee can be used in home area

networks (HAN) for smart homes and advanced metering infrastructure (AMI), which are systems that collect and analyse energy usage, communicate with smart meters. It has been declared as the most suitable communication standards for smart grid residential network domain by the U.S. National Institute for Standards and Technology (NIST) [55]. ZigBee Smart Energy Profile 2 (SEP) is an application protocol standard, which has been developed for smart grid solutions [56]. It has advantages for gas, water and electricity utilities, such as load control and reduction, demand response, real-time pricing programs, real-time system monitoring, and advanced metering support [31]. Otherwise, it is much like ZigBee itself.
ZigBee devices require very low data rate, 250 KB/s [57]. Using ZigBee protocol has also downsides, such as low processing capabilities, small memory size, small delay requirements and interference with other applications sharing the same transmission medium, license free industrial, scientific and wireless local area network frequency band from IEEE 802.11 standards, Wi-Fi, Bluetooth and Microwaves [31].
ZigBee and ZigBee SEP both use AES-128 with CCM. ZigBee uses its own network stack. ZigBee is built on the Physical layer (PHY) and the Medium Access Control layer (MAC), both defined by the IEEE 802.15.4-2003 standard. The MAC layer controls access to the radio channel using a CSMA-CA mechanism. Upon this structure, ZigBee builds the Network layer (NWK) and the Application layer (APL) which consists of the Application Support sublayer (APS) and the ZigBee Device Object (ZDO). [58]
ZigBee SEP authentication uses EAP-TLS authentication Appendix 12. [56]

4.2.3.6 Thread

Thread is relatively new protocol, only to have been announced in 2015. Thread intends to consolidate IoT protocols by working with other IoT alliances. Thread is based on open standards such as IEEE 802.15.4, IPv6, 6LoWPAN , wireless mesh personal area network [59]. This network operates at 2.4 GHz and has a data rate up to 250 kbps, has a maximum range of 30m per hop, with a default hop limit of 36 hops with the ability to connect over 250 devices in a single network [59]. It introduces convenient way to build low power networks with direct access to the Internet. Thread nodes use IPv6 to communicate with external servers, clouds, user computer or mobile device. [60]
Basic Thread topology consists of three classes : border routers, routers and end devices. Thread has a border router to be a gateway between wireless connection and LAN. Thread authentication procedure is described in Appendix 7. In this method, evaluation is done based on Joiner–Joiner Router/Commissioner sequence.

4.2.3.7 Z-Wave

Z-Wave, originally named ZenSys, is a proprietary IoT protocol, owned by Sigma Designs [59]. It uses wireless mesh network and has an operating frequency 908,42 MHz (USA) or 868,42 MHz (EU) with the data rate of 9,62 Kbps and uses frequency shift keying (FSK) data modulation [59]. Z-Wave has a maximum range of 30m a hop, with maximum of 4 hops with maximum number of devices in personal area network of 232. It uses its own smart hub to connect to the Internet [59].
Z-Wave uses collision avoidance (CA) technique for medium access control, that allows transmission of the frame when the channel is free. The data is split into 8-bit frames. Transfer layer administrates connection between two sequential layers, including retransition, checksum screening and ACK for the successful connection. Four fundamentals are used: single cast frame pattern, transfer acknowledge frame pattern, multicast frame pattern and broadcast frame pattern. Application layer is responsible for distributing the frame payload, decoding and performing commands. [61]
All devices in Z-Wave personal area network have specific identification numbers, smart hub has with the length of 32 bit, called Home-ID and Internet of Things (IoT) devices have 8 bit, called Node-ID. ID-s are received after devices are connected in the network. This is done once. Smart hub's ID is embedded into the device during the manufacturing. Z-Wave's connection to the Internet is encrypted with AES-based pre-shared key and uses 1.1 TLS [62]. Z-Wave accomplishes secure key exchange using Elliptic Curve Diffie-Hellman (ECDH) [62]. [63]
Z-Wave authentication has been described in Appendix 6.

4.2.3.8 Wi-Fi

Wireless fidelity, more widely known Wi-Fi includes IEEE 802.11abg standards for wireless local area networks. Wi-Fi allows connection to the Internet when user connects to access point or in ad hoc network. Wi-Fi uses 2.4 GHz or 5 GHz frequency band, with maximum signal rate of 54 MB/s. It has a nominal range of 100 meters, uses frequency shift keying modulation or coded orthogonal frequency division multiplexing or complementary code keying. Wi-Fi supports RC4 stream cipher WEP (wired equivalent privacy) and AES block cipher encryption , WPA2 (Wi-Fi protected access) authentication, 32-bit redundancy check (CRC) data protection. [50] WEP is old and not used any more, WPA2 is done with pre-shared key, nowadays it is done with AES.

Wi-Fi uses media access control's protocol CSMA/CA, which is connectionless and contention based.

Protocol	Frequency	Channel Width	MIMO	Maximum data rate
802.11ax	2.4 or 5GHz	20, 40, 80, 160MHz	Multi-user (MU-MIMO)	2.4 Gbps
802.11ac wave2	5 GHz	20, 40, 80, 160MHz	Multi-user (MU-MIMO)	1.73 Gbps
802.11ac wave1	5 GHz	20, 40, 80MHz	Single User (SU-MIMO)	866.7 Mbps
802.11n	2.4/5 GHz	20, 40MHz	Single User (SU-MIMO)	450 Mbps
802.11g	2.4 GHz	20 MHz	N/A	54 Mbps
802.11a	5 GHz	20 MHz	N/A	54 Mbps
802.11b	2.4 GHz	20 MHz	N/A	11 Mbps
802.11	2.4 GHz	20 MHz	N/A	2 Mbps

Table 1: IEEE 802.11 Wi-Fi protocol summary [12]

Since the invention of Wi-Fi in the 1990s, wireless networks have used several different security protocols. Each new standard provided greater security, and each promised to be easier to configure than those that came before. All of them, though, retain some inherent vulnerabilities. In addition, as each new protocol was released some systems were upgraded, and some were not. As a result, today there are a number of different security protocols in use. Some of these provide a pretty good level of protection, while some don't. Wi-Fi has four main security protocols in use today – WEP, WPA, WPA2 and WPA3.

WEP Wired Equivalent Privacy (WEP) was the first widely spread Wi-Fi security standard, and was approved for use way back in 1999. Though, as its name suggests, it was supposed to offer the same level of security as wired networks, it did not. A number of security issues were quickly found, and despite many attempts to patch them, this standard was abandoned by the Wi-Fi Alliance in 2004. WEP was introduced as part of the original 802.11 standard ratified in 1997. It's pretty recognizable by its key of 10 or 26 hexadecimal digits (40 or 104 bits). In 2004, both WEP-40 and WEP-104 were declared deprecated. There were 128-bit (most common) and 256-bit WEP variants, but with ever increasing computing power enable attackers to exploit numerous security flaws. This protocol is not in use for quite some time [64] and therefore not evaluated in this work. It has been updated by newer methods and is obsolete.

WPA The WiFi Protected Access (WPA) protocol was developed in 2003 as a direct replacement for WEP. It increased security by using a pair of security keys: a pre-shared key (PSK), most often referred to as WPA Personal, and the Temporal Key Integrity Protocol (or TKIP) for encryption. Though WPA represented a significant upgrade over WEP. WPA became available in 2003, and it was the Wi-Fi Alliance's direct response and replacement to the increasingly apparent vulnerabilities of the WEP encryption standard. The most common WPA configuration is WPA-PSK (Pre-Shared Key). [65] [2] The keys used by WPA are 256-bit, a significant increase compared to the 64-bit and 128-bit keys used in the WEP system. WPA included message integrity checks and the TKIP, which employs a per-packet key system that was radically more secure than the fixed key system used by WEP. The TKIP encryption standard was later superseded by Advanced Encryption Standard (AES). [64] [66]. TKIP is still used, but it's considered obsolete after being replaced by CCMP in 2009. [66]
WPA authentication has been described in Appendix 11.

WPA2 WPA2 was developed in 2004 as the first truly new security protocol since the invention of Wi-Fi. The major advance made by WPA2 was the usage of the Advanced Encryption System (AES), a system used by the US government for encrypting Top Secret information. At the moment, WPA2 combined with AES represents the highest level of security typically used in home WiFi networks, though there remain a number of known security vulnerabilities even in this system. WPA2 replaced WPA. Certification began in September, 2004 and from March 13, 2006 it was mandatory for all new devices to bear the Wi-Fi trademark. Most important upgrade is mandatory use of AES algorithms (instead of previous RC4) and the introduction of CCMP (AES CCMP, Counter Cipher Mode with Block Chaining Message Authentication Code Protocol, 128 Bit) as a replacement for TKIP (which is still present in WPA2, as a fallback system and WPA interoperability). [66] WPA2-PSK (Pre-Shared Key) requires a single password to get on the wireless network. It's generally accepted that a single password to access Wi-Fi is safe, but only as much as you trust those using it. [65] [2] WPA2-Enterprise - Deploying WPA2-Enterprise requires a RADIUS server, which handles the task of authenticating network users access. The actual authentication process is based on the 802.1X policy and comes in several different systems labelled EAP. Because each device is authenticated before it connects, a personal, encrypted tunnel is effectively created between the device and the network. WPA2-Enterprise uses EAP-TLS, EAP-TTLS/PAP and PEAP-MSCHAPv2 authentication protocols. [67] EAP-PSK has been described in Appendix 11 EAP-TLS in Appendix 12, EAP-TTLS Appendix 13 in and PEAP-MSCHAP in Appendix 9.

WPA3 In January 2018, the Wi-Fi Alliance announced WPA3 as a replacement to WPA2. The new standard uses 128-bit encryption in WPA3- Personal mode (WPA-PSK, pre-shared key) or 192-bit in WPA3 – Enterprise (RADIUS authentication server). WPA3-Personal modes are defined as follows [68]:

1. WPA3-Personal only Mode
2. WPA3-Personal transition Mode

WPA3-Enterprise modes are defined as follows:

1. WPA3-Enterprise only Mode - When a BSS (basic service set) is configured in WPA3-Enterprise only mode, PMF shall be set to required (MFPR(Management frame protection capable) bit in the RSN (Robust Security Network). Capabilities field shall be set to 1 in the RSNE(RSN element) transmitted by the AP(access point)), A WPA3-Enterprise STA shall negotiate PMF when associating to an AP using WPA3-Enterprise only mode.
2. WPA3-Enterprise transition Mode - When WPA2-Enterprise and WPA3-Enterprise transition Mode are configured on the same BSS, PMF shall be set to capable (MFPC bit shall be set to 1, and MFPR bit is by default set to 0 in the RSN Capabilities field in the RSNE transmitted by the AP), A WPA3-Enterprise STA shall negotiate PMF when associating to an AP using WPA3-Enterprise transition mode.
3. WPA3-Enterprise 192-bit Mode - WPA3-Enterprise 192-bit Mode may be deployed in sensitive enterprise environments to further protect Wi-Fi networks with higher security requirements such as government, defence, and industrial.

When operating in WPA3-Enterprise 192-bit Mode [69]:

1. When WPA3-Enterprise 192-bit Mode is used by an AP, PMF shall be set to required (MFPR bit in the RSN Capabilities field shall be set to 1 in the RSNE transmitted by the AP).
2. When WPA3-Enterprise 192-bit Mode is used by a STA, PMF shall be set to required (MFPR bit in the RSN Capabilities field shall be set to 1 in the RSNE transmitted by the STA).

Permitted EAP (extensible authentication protocol) cipher suites for use with WPA3-Enterprise 192-bit Mode are [69]:

1. TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 - ECDHE and ECDSA using the 384-bit prime modulus curve P-384
2. TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 - ECDHE using the 384-bit prime modulus curve P-384 - RSA $\geq$ 3072-bit modulus
3. TLS_DHE_RSA_WITH_AES_256_GCM_SHA384-RSA $\geq$ 3072-bit modulus - DHE $\geq$ 3072-bit modulus

When a WPA3 STA needs to choose between multiple AKM(Authentication and Key Management)-s on a BSS, the STA shall select the AKM in priority order from the applicable list in the subclauses below. AKM selections not listed are out of scope of this specification.
Personal Modes [69]:

1. FT (Fast BSS transition) Authentication using SAE 00-0F-AC:9
2. SAE Authentication 00-0F-AC:8
3. FT Authentication using PSK 00-0F-AC:4
4. PSK using SHA-256 00-0F-AC:6
5. PSK 00-0F-AC:2

Enterprise Modes [69]:

1. FT Authentication using IEEE Std 802.1X (SHA 256) 00-0F-AC:3
2. Authentication using IEEE Std 802.1X (SHA256) 00-0F-AC:5
3. Authentication using IEEE Std 802.1X 00-0F-AC:1

A WPA3 STA shall perform server certificate validation when using EAP-TTLS, EAP-TLS, EAP-PEAPv0 or EAP-PEAPv1 EAP methods. A WPA3 STA shall, when performing an EAP exchange with one of the above EAP methods, determine that server certificate validation has failed if none of the following are true:

1. The STA is configured with EAP credentials that include a server certificate that is exactly equal to the certificate in the received Server Certificate message.

2. The STA is configured with EAP credentials that explicitly specify a CA root certificate that matches the root certificate in the received Server Certificate message and, if the EAP credentials also include a domain name, it matches the domain name of the certificate in the received Server Certificate message.

3. The STA is configured with EAP credentials that include a domain name (FQDN or suffix-only) that matches the domain name of the certificate in the received Server Certificate message, and the root certificate of that certificate is present in the STA's trust root store.

The standards that define each EAP method specify additional conditions under which server certificate validation is required to fail. If a WPA3 STA's validation of a server certificate fails during an EAP exchange with EAP-TTLS, EAP-PEAPv0 or EAPPEAPv1, the STA shall not enter into Phase 2 of the EAP exchange.
WPA3-PSK - To improve the effectiveness of PSK, updates to WPA3-PSK offer greater protection by improving the authentication process. A strategy to do this uses Simultaneous Authentication of Equals (SAE) to make brute-force dictionary attacks far more difficult for a hacker. This protocol requires interaction from the user on each authentication attempt, causing a significant slowdown for those attempting to brute-force through the authentication process.
WPA3-Enterprise : A significant improvement that WPA3-Enterprise offers is a requirement for server certificate validation to be configured to confirm the identity of the server to which the device is connecting.
EAP-PSK has been described in Appendix 11 EAP-TLS in Appendix 12, EAP-TTLS Appendix 13in and PEAP-MSCHAP in Appendix 9.

4.2.3.9 Ultra-Wide Band

Ultra-Wide Band (UWB) is standardized over IEEE 802.15.3. Its bandwidth is over 110 Mbps (up to 480 Mbps) and can act as a wireless cable replacement of high-speed serial bus. UWB uses 3.1-10.6 GHz frequency band and its nominal range is 10 meters. It uses binary shift keying (BSK) channel modulation and supports advanced encryption standard (AES) block cipher encryption, cipher block chaining message authentication code (CBC-MAC) authentication and 32-bit cyclic redundancy check (CRC) data protection. [50] [70]
Ultra-wideband (UWB) has emerged as a technology that offers great promise to satisfy

the growing demand for low cost, high data rate, short range wireless transmission systems such as digital wireless indoor and home networks provide easy connection and efficient media exchange. UWB presents a unique opportunity to become a widely adopted radio solution for wireless personal networking technology because of the enormous bandwidth available, the potential for high data rates, and the prospect of small size and low power requirements along with low implementation cost. UWB radio transmission can legally operate in the range 3.1 to 10.6 GHz at a transmitter power of -41.3 dBM/MHz. [71] [72] The use of UWB technology under the FCC guidelines can provide huge capacity over short ranges. Currently, UWB is able to support various data rates, ranging from 110 to 480 Mbps, over distances up to 10 meters. he basic idea of UWB can be traced back to the first wireless communication system in the late 1890s. However the main concept of UWB was developed only in early 1960s through research in time-domain electromagnetic systems, where impulse measurement techniques were used to characterise the transient behaviour of a certain classes of microwave networks. Similar to spread spectrum or code division multiple-access (CDMA), UWB technology was firstly used in a military environment and just recently introduced in the commercial market. Today, UWB has been considered as one of the most promising candidates for wireless communications within a short-range RF environment and has been creating a lot of interest from research community worldwide. [71] [72] UWB authentication has been described in Appendix 17 and CCM (CBC-MAC) authentication has been shown in Appendix 18.

4.2.3.10 WiMAX

WiMAX (Worldwide Interoperability for Microwave Access) is based on IEEE 802.16 standard, which speeds as high as 70 MB/s and a range of up to 48 kilometres. WiMAX can be used for wireless networking like Wi-Fi. It uses time division multiplexing (TDM) or frequency division duplexing (FDD). WiMAX technology operates in between 2 and 11 MHz frequency range and uses power band profiles from 100 Mw up to 2W. Data rates can reach 2b/Hz. WiMAX connection supports EAP (extensible authentication protocol) and DES (data encryption standard). [73] The WiMAX network is a wireless network technology and as such its operating principle can be related and compared to the ISO OSI reference model. As a technology, which utilizes radio wave as a transmission medium, it spans over two bottommost layers of the ISO model: the physical layer and the Medium Access Control (MAC) layer.

The security architecture has been defined in a dedicated Privacy Sublayer (PS)to ensure appropriate level of security for the parties involved in a transmission. This sublayer ensures parties' authentication as well as transmitted data integrity and confidentiality. Security Association (SA) is a container of key information utilized for ensuring secure communication between a Subscriber Station (SS) and a Base Station (BS). There are two types of SA: Data SA and Authorization SA. Data SA protects communication between one or more SS-s and a BS. Data SA contains 16-bit SA identifier (SAID), Encryption cipher to protect the data exchanged over the connection, Two TEKs: one for current operation and another when the current key expires, Two 2-bit key identifiers, one for each TEK, TEK lifetime - minimum value is 30 min and the maximum value is 7 days (default is 2 days), Initialization vector for each TEK, Data SA type indicator (primary, static, dynamic). [73] Authorization SAs contain : [74]

1. X.509 certificate identifying the subscriber station.
2. 160-bit authorization key.
3. 4-bit authorization key identifier.
4. Authorization key lifetime. The minimum value is 1 day and the value maximum is 70 days (default is 7 days).
5. Key encryption key (KEK) for distributing TEKs.
6. Downlink hash function-base message authentication code (HMAC) key.
7. Uplink HMAC keys.
8. List of authorized data SA-s.

These are shared by the base station and the subscriber station. They are used by base stations to configure Data SA intended for a subscriber station. [5] Data confidentiality is ensured by symmetric Data Encryption Standard (DES), by Triple DES (3DES), Advanced Encryption Standard (AES) and asymmetric Rivest, Shamir, Adleman (RSA) algorithms. In order to ensure integrity of transmitted data, Keyed-Hash Message Authentication Code (HMAC) and Cipher-Based Message Authentication Code (CMAC) mechanisms are used. The authorization and authentication processes are implemented based on the Privacy Key Management (PKM) protocol which uses asymmetric encryption as well as on public key certificates. IT is also involved in the key management mechanism which performs as an immediate consequence of a device logging on to the network and SS authentication. [5] This protocol is based on the so-called Security Associations (SA). This is a state specific and unique for each connection, describing its cryptographic properties, such as values and validities of used cryptographic keys and used algorithms. There are two versions of the PKM protocol. The PKMv1 is used to protect nomadic networks (including LOS and NLOS connections), whereas the other – PKMV2 is used to protect WiMAX networks with mobility support (IEEE 802.16e). [5] WiMAX has two authentication protocols, PKMv1 and PKMV2. [75] WiMAX authentication has been described in Appendix 5.

4.2.3.11 6LOWPAN

The 6LoWPAN is a protocol based on the IPv6 protocol. It is designed to be used over the IEEE 802.15.4 standard for low power wireless communication. The issue with this standard is that the frames are limited to 127 bytes, including the MAC header of 23 bytes and an optional AES encryption header of 21 bytes. With a conventional IPv6 protocol, the remaining payload is reduced to 33 bytes for UDP and 21 bytes for TCP. [76] 6LOWPAN supports as well AES-128. Low-power, IP-driven nodes and large mesh network support make this technology a great option for Internet of Things (IoT) applications. As the full name implies – "IPv6 over Low-Power Wireless Personal Area Networks" – 6LoWPAN is a networking technology or adaptation layer that allows IPv6 packets to be carried efficiently within small link layer frames, such as those defined by IEEE 802.15.4. [77]
6LOWPAN supports EAP-TLS, which has been described in Appendix 12. [78]

4.3 Wired

Data transmissions are broadcast in nature for wired connections, hence, the security aspects are critical. Confidentiality, authentication, integrity, and user intervention are some of the critical issues in smart grid communications. [31]

4.3.1 Ethernet

Digital subscriber line (DSL) is high-speed digital data transmission technology that uses voice telephone network. There are two types of digital subscriber lines – symmetric (SDSL) and asymmetric (ADSL). ADSL differs from SDSL in the matter of bandwidth and bit rate towards the customer, it is higher. Greater than 1 MHz frequencies are nothing spectacular for asynchronous DSL enabled telephone lines [79]. This technology is already widespread, low-cost and has high-speed data transmission. DSL can have a

long downtime and may not be that reliable for critical infrastructure. It requires cables for communication, which must be installed and regularly maintained.
Fibre-optic is a communication type in telecommunications, which uses cables made from thin transparent glass or plastic, for example to transmit telephone signals, Internet connection and cable television signals. Broadband network access using fibre-optics is called passive optical network. There are different techniques to transmit bandwidth using fibre-optic cables.
Comparing DSL is much slower, but cheaper because of in place telephone lines and poles. Passive optical network can be extended further from the provider and it does not rely on electricity. Ethernet operates in the data link layer and the physical layer. It is a family of networking technologies that are defined in the IEEE 802.2 and 802.3 standards. Ethernet supports data bandwidths of: 10 Mb/s, 100 Mb/s, 1000 Mb/s (1 Gb/s), 10,000 Mb/s (10 Gb/s), 40,000 Mb/s (40 Gb/s) and 100,000 Mb/s (100 Gb/s). Ethernet standards define both the Layer 2 protocols and the Layer 1 technologies. For the Layer 2 protocols, as with all 802 IEEE standards, Ethernet relies on the two separate sublayers of the data link layer to operate, the Logical Link Control (LLC) and the MAC sublayers.

4.3.1.1 G.hn

Developed by ITU-T to be a unified interoperable standard consisting of all types of existing cabling home phone line, power line, coaxial cable and Cat-5 targeted to work in a residential environment as well as in small and medium offices. could provide a data rate of up to 1Gbps. G.hn can interconnect a maximum of 250 devices in one network [80]. [39] G.hn supports AES and uses password-authenticated key agreement protocol that ensures mutual authentication of two parties by using a Diffie–Hellman key exchange. [81] G.hn – stands for “next generation home network technology”– is one of several approaches to home networking that seek to make first meter connectivity easier to handle. It is a unified standard [81] for wired in-home networking developed by the International Telecommunications Union Telecom (ITU-T) Standardization Sector G.hn allows networking of all types of digital media over unshielded telephone lines, power lines and coaxial cable – the most common wires found in today’s homes. G.hn is compatible with the two most widely used Ethernet home networking technologies, wired and wireless LAN connections. G.hn allows up to 250 nodes operating in the network. It defines several profiles to address applications with significantly different implementation complexity. High-profile devices, like gateways, are capable of providing very high throughput and sophisticated management functions. Low-profile devices, such as home automation, have low throughput and basic management functions but can interoperate with higher profiles. [82] G.hn authentication has been described in Appendix 3.

4.3.1.2 LonTalk Protocol

LonTalks is a protocol for LonWorks platform, which was created for the control of applications. LonTalks is defined by standard ISO/IEC 14908.1. It supports network connections over twisted pair, powerlines, fibre optics, and RF. It was developed by Echelon Corporation. Its maximum data rate is 1,25 MB/s. LonTalk does not really use encryption, rather encodes, where any 48-bit key is a valid encryption key. [83]
LonTalk protocol is designed for communication in control networks. These networks are characterized by short messages, very low per node cost per multiple communications media, low bandwidth and maintenance, multi-vendor equipment and low support costs. [84]
Authentication protocol has two asymmetric parts, the challenger and the challengee. The authentication process is initiated by the challenger, which generates a random number X; next, the challengee responds with Y = E(X msg), an encryption of X and the original message using a private key; and finally the challenger compares Y with its own version of E(X msg) and makes a pass/fail decision based on the outcome. [83]
LonTalk authentication has been described in Appendix 16.

4.3.2 Powerline Communication

Powerline communication (PLC) is a technique, which uses existing powerlines to transmit data signals from one device to another. Due to the direct contact of the electricity meter it is the first choice for communication [85]. PLC technology sets restrictions for applications, because it has low bandwidth network, 20 kb/s for neighbourhood area networks [33]. For PLC, the network topology needs the devices to be connected to the powerlines, wiring distances can be huge and. This affects the quality of the broadcasted signal, therefore powerlines are not suited for data transmission. However, there are hybrid solutions with different techniques, for example GPRS, GSM to provide full connectivity [31].

4.3.2.1 Universal Powerline Bus

UPB is a proprietary software protocol developed by Powerline Control Systems. UPB communication is a method of reliably communicating command, control, and status information across an electrical AC powerline. The UPB powerline communication method consists of transmitting digitally encoded information over the electrical powerline as a series of precisely timed electrical pulses (called UPB Pulses) that are superimposed on top of the normal AC power waveform (sine wave). Receiving UPB devices can easily detect and analyse these UPB Pulses and pull out the encoded digital information from them. [86] UPB messages are limited to 480 bps. The UPB addressing scheme allows for 250 systems (houses) on each transformer and 250 devices on each system. UPB Pulse is capable travelling large distances over the powerline and even coupling through the power transformer to the other side of a split phase power arrangement. [86] Each UPB device must be programmed with a Unit ID. The unit ID will have a value between 1 and 250. Units out of the box from the factory will have a Unit ID assigned that equals the manufacturer's product ID. [87]

Each UPB product must also have a Network ID. When control signals are sent out on the powerline, part of the signal is the Network ID. Only devices that have that specific Network ID will respond to the signal and take the appropriate action. In real life, it is expected that a Network ID will correspond to one home. Adjacent homes should use different Network IDs, in order to prevent signals in one home from controlling devices in the next door house. The appropriate range for Network IDs is 1 – 250, with the default Network ID out of the box being FF (hex) or 255 decimal. [87]

Each UPB product must also have a Network Password. In order to program a device or change its programming, you must know the appropriate Network ID, and then the appropriate Network Password. If two homes side by side have the same Network ID, but different passwords, then users in both homes will be able to control devices in each home, but they can't program devices in the other home, only their own. Again, this highlights the importance of using different Network IDs for adjacent homes. The Network Password is a four character alpha-numeric password, with each character allowed to be in the range of 0 – F (hexadecimal). The default Network Password out of the box is 1234 (hex). [87]

While the network name has little importance, it is a means of determining which network is being used, especially in the case of a multi-network system. The default network name out of the box is New Network Name. However, the Network Name is not used as a primary means of identification, and has little importance. UPB devices are also programmed for a room name, and a device name. The room name is important within various controller environments, as it organizes all devices according to rooms. However, it is not important for manual setup. The device name merely gives the user the ability to name a device so that it is easily understood what the device does. For example, rather than being named New Lamp Module, the module can be named bedside table lamp [87]

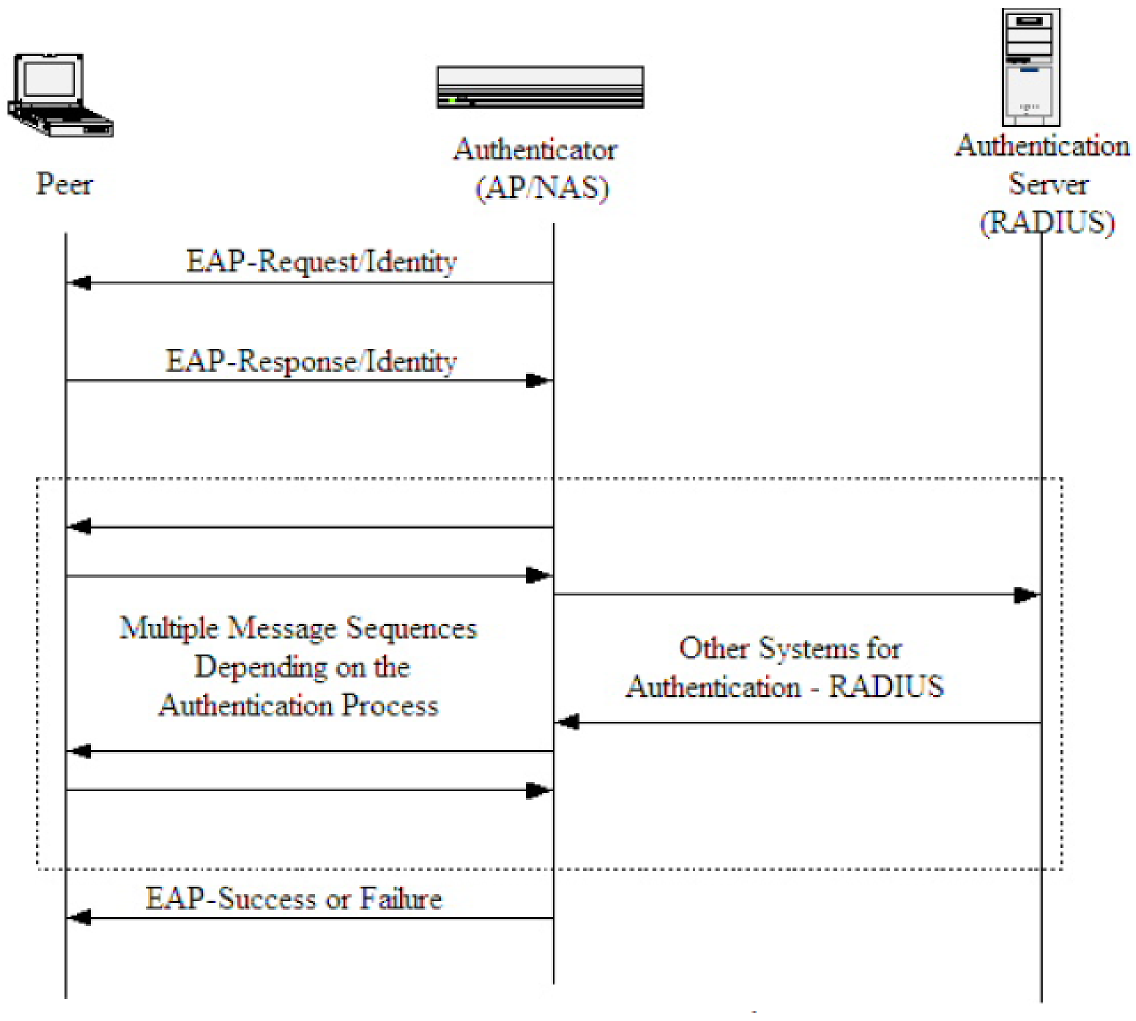
4.4 Hybrid

4.4.1 Extensible Authentication Protocol

Extensible Authentication Protocol (EAP) is an authentication framework providing multiple authentication methods, implemented as an arbitrary authentication method for a network access connection. [2] EAP typically is used over data link layers such as Point-to-Point Protocol (PPP) or IEEE 802, without requiring IP. EAP provides its own support for duplicate elimination and retransmission, but is reliant on lower layer ordering guarantees. [88] EAP was originally proposed for the Point-to-Point Protocol for an optional authentication phase after the PPP link has been established. It is also a general purpose authentication protocol. EAP supports multiple authentication methods, such as token card, Kerberos, one-time password, certificate, public key authentication, and smart card. [2] EAP supports wireless and wired connections. For example TLS and TTLS and PEAP authentication methods use EAP. [88]
When using EAP, it is not necessary to pre-negotiate a particular authentication mechanism at the Link Control Phase. Instead, the authenticator usually sends an initial Identity Request followed by one or more Requests to authenticate the supplicant. A Request contains a type field to indicate what information is being requested. The supplicant then replies a Response for each Request. The Response also contains a type field according to the type field in the Request. Based on the specific authentication mechanism, a series of Requests and Responses will be exchanged. The authenticator then either sends an authentication Success or Failure to the supplicant. [89]
Message exchange for generic Extensible Authentication Protocol is show in Figure 1:

Authentication methods of EAP-PSK has been described in Appendix 11 EAP-TLS in Appendix 12, EAP-TTLS Appendix 13in and PEAP-MSCHAP in Appendix 9.

Figure 1: EAP message flow for authentication [2]


Peer
Authenticator (AP/NAS)
Authentication Server (RADIUS)
EAP-Request/Identity
EAP-Response/Identity
Multiple Message Sequences Depending on the Authentication Process
Other Systems for Authentication - RADIUS
EAP-Success or Failure

4.4.2 KNX Protocol

KNX is open standard (EN 50090, ISO/IEC 14543) for home automation. Technology provides manufacturer and application domain independent KNX bus that interconnect devices that can support twisted pair, power line, RF and IP communication in an integrated manner. KNX supports implementation of AES and Diffie-Hellman algorithms for encryption and authentication [90]. At the end of 2003, the KNX Standard was approved by CENELEC (European Committee of Electrotechnical Standardisation) as the European Standard for Home and Building Electronic Systems as part of the EN 50090 Series. The KNX Standard was also approved by CEN (EN 13321-1 for media and protocol and EN 13321-2 for KNXnet/IP). At the end of 2006, KNX was also approved as a world standard (ISO/IEC 14543-3). In 2007, the Chinese translation of the international standard achieved GB/Z status as GB/Z 20965. KNX is also approved in the USA as ANSI/ASHRAE 135. Due to the flexibility of the KNX technology, a KNX installation can easily be adapted to the changing circumstances of the user. Various communication media (and hence transmission methods) can be used for the exchange of data between devices in a KNX system: [91]

1. KNXTwistedPair (KNXTP) – communication via a twisted pair data cable (bus cable)
2. KNX Powerline (KNX PL) – uses the existing 230 V mains network
3. KNX Radio Frequency (KNX RF) – communication via radio signal
4. KNX IP – communication via Ethernet

In KNX TP the bus cable supplies all bus devices with both data and power. The rated voltage of the bus system is 24 V, while the voltage provided by the power supplies is 30 V. The data transfer rate is 9,600 bit/s, and the data travel serially, one byte at a time, via asynchronous data transfer. Access to the KNX bus, like several other bus systems, is random and event-driven. A telegram can only be transmitted if no other telegram is being transmitted at the same time. [91] KNX Authentication has bee shown in Appendix 4.

4.4.3 X10

X10 was invented in 1975 by Pico Electronics in Glenrothes, Scotland. It is one of the oldest protocols in the market. It is unreliable, slow and system is difficult to install. This protocol does not support encryption. X10 was originally designed for powerline communication only but through time the need for wireless communication emerged. X10 signal is composed of a series of 5-volt, 121 kHz pulses having a duration of 1 millisecond, positioned at zero crossings of the 60 Hz AC power signal. Each pulse corresponds to a binary 1, and the absence of a pulse corresponds to a binary 0. A single X10 command consists of a 22-bit word obtained from eleven complete cycles of the AC power signal. [92]
X10 Technology is 20 years old, it was first meant for controlling low cost lighting and appliance control devices. X10 powerline technology transmits binary data using AM technique. X10 enables control over lights and virtually any other electrical device from anywhere in the house with no additional wiring, just with a controller or transmitter in electrical outlet. The controller/transmitter could use the electrical wiring as the transmission media to communicate with those modules. X-10 powerline technology employs an Amplitude Modulation (AM) technique to transmit binary data. [93]
X10 technology is still in use but there is almost no documentation for X10 protocol. Unfortunately, this very simple and convenient technology does not support any authentication or encryption at all.

4.4.4 Insteon

Insteon uses wireless and powerline connection and merges these connection technologies into a single network. It operates over 915 MHz/ 869.85 MHz/921.0 MHZ wireless radio band on a peer-to-peer network with the data rate of 180 bps. Wireless range reaches to 45 meters and the power line uses 131.65 KHz. [39]
Insteon network security is maintained at two levels. All-Linking Control ensures that users cannot create All-Links that would allow them to control their foreign Insteon devices, even though those devices may be repeating each other's messages. Encryption within extended-length messages permits completely secure communications for applications that require it. Insteon enforces ALL-Linking Control by requiring that users have physical possession of devices in order to actually create All-Links, and by masking Non-linked Network Traffic when messages are relayed outside the Insteon network itself. [94] Firmware in Insteon devices does not allow them to identify themselves to other devices unless a user physically presses a button on the device. [94]

If owner pushes button on both the Controller device and the Responder device then an All-Link has been established between them. Responder will not act on commands from an unlinked controller. All-Linking by sending Insteon messages requires knowledge of the 3-byte addresses of Insteon devices. These addresses are unique for each device and assigned at the factory. They are printed on the labels of the devices. Codes can be read from the label and typed to management program. Insteon really relies on physical security rather than communicative mechanisms. [94] For applications such as door locks and security systems, Insteon Extended-length messages can contain encrypted payloads. Possible encryption methods include rolling-code, managed-key, and public-key algorithms. The encryption method that will be certified as the Insteon standard is currently under development. [94]

4.4.5 Wireless HART Protocol

HART is an acronym for Highway Addressable Remote Transducer, which is divided to analogue or digital industrial automation open protocol. The HART Protocol superimposes digital communication signals at a low level on top of the 4-20mA. This enables two-way field communication and makes it possible for additional information beyond just the normal process variable to be communicated to/from a smart field instrument, and there is no interference with the 4-20mA signal. Audio frequency-shift keying (FSK), which uses modulated tones to produce a digital signal, transfers the digital information containing the phone number. The data transfers at a rate of 1,200 bps using 1,200 Hz and 2,200 Hz frequencies representing a binary 1 or 0. [95] Wireless HART is a wireless mesh network communications technology for process automation applications. It adds wireless capabilities to the HART Protocol while maintaining compatibility with existing HART devices, commands, and tools. Wireless HART is built upon the IEEE 802.15.4 standard for low-power mesh radio networks. It is based on Time Division Multiple Access (TDMA). Wireless HART uses several mechanisms in order to successfully coexist in the shared 2.4GHz ISM band: Frequency Hopping Spread Spectrum (FHSS) allows Wireless HART to hop across the 16 channels defined in the IEEE 802.15.4 standard in order to avoid interference. Clear Channel Assessment (CCA) is an optional feature that can be performed before transmitting a message, the transmit power level is configurable, and a mechanism to permit the use of certain channels, called blacklisting, is available. [96]
Wireless HART uses uses IEEE 802.15.4 compatible radios operating in the 2.4GHz Industrial, Scientific, and Medical radio band. [97] Wireless HART security protocol uses CCM mode in conjunction with AES-128 block cipher using symmetric keys, for the

message authentication and encryption. A public and private keys are used to establish a secure communication. A new device is provisioned with a "join key" before it attempts to join the wireless network. The Join key is used to authenticate the device for a that specific Wireless HART network. [98] Once the device joined the network, the Network manager is able to provide it with Session and Network keys. The real key generation and management is done by a Security manager, which is not specified by Wireless HART, but the keys are distributed to the Network devices by the Network manager. A Session key is used by the Network layer to authenticate the end-to-end communication between two devices. Different Session keys are used for each pairwise communication. The Data Link layer uses a Network key to authenticate messages on a one-hop basis. A well-known Network key is used when a device attempts to join the network. [98] [96] Each Wireless HART network includes three main elements:

1. Wireless field devices connected to process or plant equipment. These devices can be a device with Wireless HART built in or an existing installed HART-enabled device with a Wireless HART adapter attached to it.

2. Gateways enabling communication between the field devices and host applications connected to a high-speed backbone or other existing plant communications network.

3. A Network Manager responsible for configuring the network, scheduling communications between devices, managing message routes, and monitoring network health. The Network Manager can be integrated into the gateway, host application, or process automation controller.

CCM authentication has been shown in Appendix 19.

5 Methodology

The main research method for this work was formal verification of protocols with ProVerif and BAN logic, but also quantitative analysis of existing data comparing and analysing different aspects of the categories used in evaluating protocol authentication was conducted. Evaluation method is proposed from the analysis and verification of these authentication protocols. Scope for selecting the protocols is most prominent protocols from different connection technologies. Proposed method is applicable to all smart home and smart grid authentication protocols. New protocols could be also developed, at least according to this method. Proposed evaluation method checks if criterion is implemented and required by default in the technology and authentication protocol. This method does not rank different cryptographic algorithms or evaluate key lengths, it is for checking if there is a mechanism or answer to this criterion. These protocols are verified with ProVerif [99] and using BAN-Logic [17]. For evaluation method not only security criterion were considered. Criterion derived from the analysis were split to 5 different categories: transport, administration, audit, efficiency and security. Different criterion are derived from previous proposed standards, best practices and feasibility. Scoring is done by the multi-attribute utility (MAU) analysis and paired comparison.

5.1 Requirements

Requirements for this method have been chosen from previous protocol standards and proposed methods, protocol enhancements as well as for device compliance.

Transport

1. Independence - Authentication protocol transport must be independent, it must be compliant to be transported on any given transport protocol. [100]

2. Congestion Control - Protocol must have method for congestion control, transmission control or medium access control over shared and public channels. [101]

3. Message Matching - Criteria for checking if mechanism for matching messages MAC or IP address based. [101]

4. IP - Criteria for IP4 and IP6 support. [101], [102]

Administration

1. Data Types - All types of data should be supported, integer, characters, as well as unified character set agreed on, to prevent compliance issues. [100] [103]
2. Request and Response Management - Requests and Responses must be kept count of, as well as responded. Value is true if it is implemented in that protocol and technology. [104]

Audit

1. Message Headers - Messages must have headers for identification and verification. [101]
2. Accounting Management - For checking if any user authentication in present. [101]

Efficiency

1. Scalability - Value is true, if at least mesh network communication is supported. [100]
2. Constrained - For evaluating if protocol is suited for communication with low-power narrowband devices. [104]

Security

1. Authentication Cluster - Criteria for evaluating, if multiple authentication server presence is supported and therefore switching authentication authorities to prevent authentication failure.
2. Error Detection and Correction - Criteria for any packer or frame error detection and correction mechanism. [104], [105]
3. Encapsulation - Indicator for payload encapsulation support. [100]
4. Encryption - Value is TRUE, if any kind of encryption mechanism is implemented by default. [100], [102]
5. Certificates - Value is TRUE if user, client or server certificate based authentication is supported. [106] [104]

6. Integrity Check - Indicator for frame or packet integrity mechanism presence. [104], [105]
7. Mutual Authentication - Criteria for detecting mutual authentication. [100], [105]
8. Tunnelling - Tunnelling protocol support for private network communications. [100] [107]
9. Reachability - ProVerif verification for event executions and injections, value is TRUE if attacker is not able to inject nor execute events with its own variables. [108]
10. Secrecy - ProVerif verification for decrypted variable and private key, shared key and passwords secrecy, value is TRUE if attack is not able to reach to this value. [100], [108]
11. Crypto-Agility - Support for multiple cryptographic algorithms or ciphers for encryption.
12. Authentication Authority - Presence of at least authoritative authentication or checking device in the network.
13. Key Length - Value is TRUE, if the authentication protocol key length is greater or equal than recommended size in [109].
14. Authentication Synchronization - Value is TRUE if authentication is done within time window or period, there for connections are synchronized during authentication. [108]
15. Authenticated Access - Value is TRUE, if the device is authenticated joining the network. [108], [102]
16. Authentication Logic - Verification done according to BAN logic. Result is TRUE if the postulated goal has been reached. [108]

5.2 ProVerif

Goal of ProVerif is the verification of cryptographic protocols. Cryptographic protocols are concurrent programs which interact using public communication channel such as the Internet to achieve some security-related objective. Since the attacker is assumed to have a complete control of the communication channels, the attacker may: read,

modify, delete, and inject all messages. The attacker is also able to manipulate data, for example: compute the element of a tuple and decrypt messages if it has the necessary keys. ProVerif is able to capture the behaviour of adversaries. Only protocol must be modelled to invoke these rules. ProVerif's input language allows such cryptographic protocols and associated security objectives to be encoded in a formal manner, allowing ProVerif to automatically verify claimed security properties. [110]
In ProVerif cryptography is assumed to be perfect; that is, the attacker is only able to perform cryptographic operations when in possession of the required keys. In other words, it cannot apply any polynomial-time algorithm, but is restricted to apply only the cryptographic primitives specified by the user. The relationships between cryptographic primitives are captured using rewrite rules and/or an equational theory. [110]
The ProVerif tool is able to prove reachability properties, correspondence assertions, and observational equivalence. Proving reachability properties is ProVerif's most basic capability. The tool allows to capture of which terms are available for an attacker; and secrecy of terms can be evaluated with respect to a model.
Correspondence assertions are used to capture relationships between events which are expressed in the form "if an event e has been executed, then event e' has been previously executed." [110]
These events contain arguments, which allow relationships between the arguments of events to be studied and executed. The correspondence is insufficient to capture authentication in cases where a one-to-one relationship between the number of protocol runs performed by each participant is desired. In order to understand the results correctly, it is important to understand the difference between the attack derivation and the attack trace. The attack derivation is an explanation of the actions that the attacker has to make in order to break the security property, in the internal representation of ProVerif. [110]

ProVerif can display three kinds of results:

1. RESULT[Query] is true: The query is proved, there is no attack. In this case, ProVerif displays no attack derivation and no attack trace. [110]
2. RESULT [Query] is false: The query is false, ProVerif has discovered an attack against the desired security property. The attack trace is displayed just before the result (and an attack derivation is also displayed, but you should focus on the attack trace since it represents the real attack). [110]

3. RESULT [Query] cannot be proved: This is a “don’t know” answer. ProVerif could not prove that the query is true and also could not find an attack that proves that the query is false. Since the problem of verifying protocols for an unbounded number of sessions is undecidable, this situation is unavoidable. Still, ProVerif gives some additional information that can be useful in order to determine whether the query is true. In particular, ProVerif displays an attack derivation. By manually inspecting the derivation, it is sometimes possible to reconstruct an attack. For observational equivalence properties, it may also display an attack trace, even if this trace does not prove that the observational equivalence does not hold. [110]

Unfortunately, no software is sound and bulletproof, ProVerif cannot fully model the incrementation of the message counter and the tests performed on this counter to accept or reject the message. Trying to prove key validity is impossible in mutual authentication lack of third party, key correctness and ”goodness” must be an assumption to validate even authentication [110]

5.3 BAN logic

Burrows, Abadi and Needham (BAN) logic is the logic of beliefs , which is based on the authentication of entities and how principals relationships evolve during the run of a protocol. It can be used to describe the exchange of messages, explaining what is needed and what must be considered. BAN logic does not consider all aspects of security protocols. This logic operates at an abstract level and therefore does not consider implementation errors or inappropriate use of cryptosystems. [111]
BAN logic is simple and because of that not a powerful tool to consider all aspects of a protocol. Verifying protocol with BAN logic means deriving the beliefs that honest principals correctly executing a protocol can come to, as a result of the protocol execution. [111]
In order to use BAN logic, protocol must be transformed to idealized form. After the protocol has been taken to idealized form, postulates of the logic and the inference rules can be applied to the formulae. The idealized protocols do not include cleartext message parts. Cleartext communication is omitted simply because it can be forged, and so its contribution to an authentication protocol is mostly one of providing hints as to what might be placed in encrypted messages.
Not all clear text authentication protocols messages could be omitted in this work.
In order to analyse idealized protocols, messages are annotated with logical formulas,

much as in a proof in Hoare logic [112]. Formulas are written before the first message and after each message. The main rules for deriving legal annotations are If X holds before the message P + Q: Y then both X and Q sees Y hold afterwards and if Y can be derived from X by the logical postulates then Y holds whenever X holds. [17]

$P \mid\equiv X$: P believes X, or P would be entitled to believe X. In particular, the principal P may act as though X is true. This construct is central to the logic.

$P \triangleleft X$: P sees X. Someone has sent a message containing X to P, who can read and repeat X (possibly after doing some decryption).

$P \mid\sim X$: P once said X. The principal P at some time sent a message including the statement X. It is not known whether the message was sent long ago or during the current run of the protocol, but it is known that P believed X then.

$P \Rightarrow X$: P has jurisdiction over X. The principal P is an authority on X and should be trusted on this matter. For example, a server is often trusted to generate encryption keys properly. This may be expressed by the assumption that the principals believe that the server has jurisdiction over statements about the quality of keys.

$\#(X)$: The formula X is fresh; that is, X has not been sent in a message at any time before the current run of the protocol. This is usually true for nonces, that is, expressions invented for the purpose of being fresh. Nonces commonly include a timestamp or a number that is used only once.

$P \overset{K}{\leftrightarrow} Q$: P and Q may use the shared key K to communicate. The key K is good, in that it will never be discovered by any principal except P or Q, or a principal trusted by either P or Q.

$\overset{K}{\mapsto} P$: P has K as a public key. The matching secret key (denoted K-l) will never be discovered by any principal except P or a principal trusted by P. $P \overset{X}{\rightleftharpoons} Q$:The formula X is a secret known only to P and Q, and possibly to principals trusted by them. Only P and Q may use X to prove their identities to one another. An example of a secret is a password.

$\{X\}_K$: This represents the formula X encrypted under the key K. Formally, X), is a convenient abbreviation for an expression of the form X), from P. We make the realistic assumption that each principal is able to recognize and ignore his own messages; the originator of each message is mentioned for this purpose.

$\langle X\rangle_Y$: This represents X combined with the formula Y; it is intended that Y be a secret and that its presence prove the identity of whoever utters (X) y. In implementations, X is simply concatenated with the password Y. Notation highlights that Y plays a special role, as proof of origin for X, in much the same way as an encryption key.
The message-meaning rules concern the interpretation of messages. Two of the three concern the interpretation of encrypted messages, and the third concerns the interpretation of messages with secrets. They all explain how to derive beliefs about the origin of messages.

For shared keys:

$$\frac{P \mid\equiv \overset{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \mid\equiv Q \mid\sim X}$$

That is, if P believes that the key K is shared with Q and sees X encrypted under K, then P believes that Q once said X. For this rule to be sound, it must be guaranteed that P did not send the message himself, it suffices to recall that $\{X\}_K$ stands for a formula of the form $\{X\}_K$ from R, and to require that$R \neq P$.

Similarly, for public keys, it can be postulated that

$$\frac{P \mid\equiv \overset{K}{\mapsto} Q, P \triangleleft \{X\}_{K_{-1}}}{P \mid\equiv Q \mid\sim X}$$

For shared secrets, it can be postulated that

$$\frac{P \mid\equiv Q \overset{X}{\leftrightharpoons} P, P \triangleleft \langle X\rangle_Y}{P \mid\equiv Q \mid\sim X}$$

That is, if P believes that the secret Y is shared with Q and sees $\langle X\rangle_Y$, then P believes that Q once said X. This postulate is sound because the rules for sees (given below) guarantee that $\langle X\rangle_Y$ was not just uttered by P himself.
The nonce-verification rule expresses the check that a message is recent and, hence, that the sender still believes in it:

$$\frac{P \mid\equiv \#(X), P \mid\equiv Q \mid\sim X}{P \mid\equiv Q \mid\equiv X}$$

That is, if P believes that X could have been uttered only recently (in the present) and that Q once said X (either in the past or in the present), then P believes that Q believes X. For the sake of simplicity, X must be "cleartext"; that is, it should not include any subformula of the form $\{Y\}_K$.
The jurisdiction rule states that if P believes that Q has jurisdiction over X then P trusts Q on the truth of X:

$$\frac{P \mid\equiv Q \Rightarrow X, P \mid\equiv Q \mid\equiv X}{P \mid\equiv X}$$

If a principal sees a formula, then he also sees its components, provided he knows the necessary keys:

$$\frac{P \triangleleft \langle X \rangle_Y}{P \triangleleft X}$$
$$\frac{P \triangleleft \{X\}_Y}{P \triangleleft X}$$
$$\frac{P \mid\equiv Q \overset{K}{\leftrightharpoons} P, P \triangleleft \{X\}_K}{P \triangleleft X}$$
$$\frac{P \mid\equiv \overset{K}{\mapsto} P, P \triangleleft \{X\}_K}{P \triangleleft X}$$
$$\frac{P \mid\equiv \overset{K}{\mapsto} Q, P \triangleleft \{X\}_{K_{-1}}}{P \triangleleft X}$$

Recall that$\{X\}_K$, stands for a formula of the form $\{X\}_K$, from R. As a side condition, it is required that $R \neq P$; that is, $\{X\}_K$ is not from P himself. A similar condition applies to $\{X\}_{K_{-1}}$. The fourth rule is justified by the implicit assumption that if P believes that K is his public key then P knows the corresponding secret key,K_{-1} . Note that if $P \triangleleft X$ and $P \triangleleft Y$ it does not follow that $P \triangleleft (X, Y)$, since this means that X and Y were uttered at the same time.
If one part of a formula is fresh, then the entire formula must also be fresh:

$$\frac{P \mid\equiv \#(X)}{P \mid\equiv \#(X, Y)}$$

5.4 Evaluation

In a technology evaluation, protocols must be evaluated and scored against a set of evaluation criteria in order to determine the best choice. Clear assessment criteria have been produced in Evaluation criteria paragraph.Scoring is done multiattribute utility (MAU) analysis with, within the mathematical field of decision analysis. Decision analysis is concerned with providing a mathematical framework for decision making, so that decision makers can rigorously and consistently express their preferences, in such a way that results can be readily and logically explained.
Multiattribute utility (MAU) analysis is a wellestablished decision analysis method that specifically addresses how to select one alternative from a set of alternatives, which is selecting a particular product from a set of products in a given technology area. In preparing for the evaluation testing, the first step was to establish the evaluation criteria. First step for successful evaluation was to choose criterion to determine scope of this work. Evaluation criteria is specific, Boolean, true or false, types of questions that are clearly stated and can be clearly tested. False if a product does not meet evaluation criteria, True if a product fully meets evaluation criteria. The final step was to assign weights to each criterion. These weights serve as scaling factors to specify the importance of each criterion. Because they are scaling factors that specify relative importance in the overall set of criteria, they should be non-negative numbers that sum to 1.
Paired Comparison was used for determining the weights in this method, which is ordering criteria from highest importance to least importance. In the evaluation of a security product, security is the most important category. In this method it is followed by efficiency, auditing, administration/management, and then transport. Starting with the alternative of highest importance, express its importance with the alternatives of lower importance in terms of a ¡, =, or ¿ relationship.
Following relationships of importance between categories were determined. To get the weights, following equations were calculated:

$$E(Transport) = 1$$

$$D(Administration) > E$$

$$C(Audit) = D + E$$

$$B(Efficiency) = C + D + E$$

$$A(Security) = B + C + D + E$$

Back solving these equations gave the weights for these categories:

$$Security(A) = 12$$

$$Efficiency(B) = 6$$

$$Audit(C) = 3$$

$$Administration(D) = 2$$

$$Transport(E) = 1$$

These weights are used in evaluation by multiplying True or False with the weight and summarizing to received total and then dividing the result with total sum of weight, which gives a result of a score from 0 to 1.

6 Protocol Evaluation

In the evaluation ProVerif verification could not be done for CCM, because of the limitations of the software, also skipped are protocols which have no authentication by default or did not have cryptography implementations, same for BAN logic. Since BAN logic is verifying of authentication logic and ProVerif is for proving cryptographic protocols.

Evaluation tables contain protocol criterion, their weight for calculating overall score and the result of the criterion for that particular protocol. Results have citation to a paper where it has been deducted or links to the sections where is the ProVerif verification result or BAN analysis of that protocol.

If for there was no information for criterion or not clear, TRUE value could not be given. Considering the weights in these tables, security is the most important, which are all equally scored. Next important category is efficiency, because the evaluation is about smart home and smart grid protocols. These two categories are the ones that matter the most, in this method. Smart home and smart grid protocols need to have constraints and higher scalability because of the environment and technology of these devices which are controlled.

Each criterion is described in (requirements subsection). In the evaluation tables, links are after the result referring to the paper that this result is based on, authentication logic (BAN logic), secrecy (ProVerif) and reachability (ProVerif) results are linked to the appendices of specified verification method.

6.1 Authentication Protocol Evaluation 1

Criteria	Category	Weight	2G (GSM)	3G (UMTS)	4G (LTE)
Independence	Transport	1	FALSE	FALSE	FALSE
Congestion Control	Transport	1	TRUE [113], [114]	TRUE [115]	TRUE [44]
Message Matching	Transport	1	TRUE [114]	TRUE [115], [116]	TRUE [117]
Connection Types	Transport	1	FALSE	FALSE	FALSE
IP	Transport	1	FALSE	FALSE	TRUE [44]
Data Types	Administration	2	TRUE [118]	TRUE [119]	TRUE [117]
Request and Response Management	Administration	2	TRUE [42]	TRUE [115]	TRUE [44]
Message Headers	Audit	3	TRUE [114]	TRUE [115]	TRUE [44]
Accounting Management	Audit	3	TRUE [42], [114]	TRUE [120], [116]	TRUE [120]
Scalability	Efficiency	6	TRUE	TRUE	FALSE
Constrained	Efficiency	6	FALSE [121]	FALSE [121]	FALSE [121]
Authentication Cluster	Security	12	FALSE	FALSE	FALSE
Error Detection and Correction	Security	12	TRUE [114]	TRUE [115]	TRUE [44]
Encapsulation	Security	12	FALSE	TRUE [116]	TRUE [122]
Encryption	Security	12	TRUE [123], [114]	TRUE [124], [125], [43]	TRUE [43], [120], [125]
Certificates	Security	12	FALSE	FALSE	FALSE
Integrity Check	Security	12	TRUE [126]	TRUE [126]	TRUE [126]
Mutual Authentication	Security	12	FALSE [127]	TRUE [124], [125], [43]	TRUE [43], [120], [125]
Tunneling	Security	12	FALSE	TRUE [128]	TRUE [129]
Reachability	Security	12	FALSE ProVerif	TRUE ProVerif	TRUE ProVerif
Secrecy	Security	12	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif
Authentication Authority	Security	12	TRUE [123], [114], [42], [127]	TRUE [124], [125], [43]	TRUE [43], [120], [125]
Key Length	Security	12	FALSE [123], [114]	TRUE [124], [125], [43]	TRUE [43], [125]

Continued on next page

Table 2 – *Continued from previous page*

Criteria	Category	Weight	2G (GSM)	3G (UMTS)	4G (LTE)
Authentication Synchronization	Security	12	TRUE [123], [114]	TRUE [124], [125], [43]	TRUE [43], [125]
Authenticated Access	Security	12	TRUE [123], [114]	TRUE [124], [125], [43]	TRUE [43], [125]
Authentication Logic	Security	12	TRUE (BAN logic)	TRUE (BAN logic)	TRUE (BAN logic)

Table 2: Authentication Protocol Evaluation 1

6.2 Authentication Protocol Evaluation 2

Criteria	Category	Weight	EnOcean SP3 Mutual	EnOcean SP3 Unilateral	ZigBee SEP	Thread
Independence	Transport	1	FALSE	FALSE	FALSE	FALSE
Congestion Control	Transport	1	FALSE [52]	FALSE [52]	TRUE [56]	TRUE [130]
Message Matching	Transport	1	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [7]
Connection Types	Transport	1	FALSE	FALSE	FALSE	FALSE
IP	Transport	1	FALSE	FALSE	FALSE	TRUE [104]
Data Types	Administration	2	TRUE	TRUE	TRUE [56]	TRUE [104]
Request and Response Management	Administration	2	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [104]
Message Headers	Audit	3	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [7]
Accounting Management	Audit	3	FALSE	FALSE	FALSE	FALSE
Scalability	Efficiency	6	FALSE	FALSE	TRUE [56]	TRUE [130]
Constrained	Efficiency	6	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [7]
Authentication Cluster	Security	12	FALSE	FALSE	FALSE	FALSE
Error Detection and Correction	Security	12	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [104]
Encapsulation	Security	12	TRUE [4]	TRUE [4]	FALSE	TRUE [7]
Encryption	Security	12	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [7]
Certificates	Security	12	FALSE	FALSE	TRUE [56]	TRUE [104]
Integrity Check	Security	12	TRUE [4], [52]	TRUE [4], [52]	TRUE [56]	FALSE
Mutual Authentication	Security	12	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [7]
Tunneling	Security	12	FALSE	FALSE	TRUE [56]	FALSE
Reachability	Security	12	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif
Secrecy	Security	12	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif
Crypto-Agility	Security	12	FALSE [4]	FALSE [4]	TRUE [56]	FALSE
Authentication Authority	Security	12	FALSE [4]	FALSE [4]	TRUE [56]	TRUE [7]

Continued on next page

Table 3 – *Continued from previous page*

Criteria	Category	Weight	EnOcean SP3 Mutual	EnOcean SP3 Unilateral	ZigBee SEP	Thread
Key Length	Security	12	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [7]
Authentication Synchronization	Security	12	TRUE [4]	TRUE [4]	TRUE [56]	TRUE [7]
Authenticated Access	Security	12	FALSE	FALSE	TRUE [56]	TRUE [7]
Authentication Logic	Security	12	TRUE (BAN logic)	TRUE (BAN logic)	TRUE (BAN logic)	(BAN logic)

Table 3: Authentication Protocol Evaluation 2

6.3 Authentication Protocol Evaluation 3

Criteria	Category	Weight	WPA3 (EAP-TLS)	WPA3 (EAP-TTLS)	WPA3 (PEAP-MSCHAPv2)
Independence	Transport	1	TRUE [68]	TRUE [68]	TRUE [68]
Congestion Control	Transport	1	TRUE [68]	TRUE [68]	TRUE [68]
Message Matching	Transport	1	TRUE [68]	TRUE [68]	TRUE [68]
Connection Types	Transport	1	FALSE	FALSE	FALSE
IP	Transport	1	TRUE [68]	TRUE [68]	TRUE [68]
Data Types	Administration	2	TRUE [68]	TRUE [68]	TRUE [68]
Request and Response Management	Administration	2	TRUE [68]	TRUE [68]	TRUE [68]
Message Headers	Audit	3	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Accounting Management	Audit	3	TRUE [68]	TRUE [68]	TRUE [68]
Scalability	Efficiency	6	TRUE [68]	TRUE [68]	TRUE [68]
Constrained	Efficiency	6	FALSE	FALSE	FALSE
Authentication Cluster	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Error Detection and Correction	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Encapsulation	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Encryption	Security	12	TRUE [69]	TRUE [69]	TRUE [69]
Certificates	Security	12	TRUE [69]	TRUE [69]	TRUE [69]
Integrity Check	Security	12	TRUE [69]	TRUE [69]	TRUE [69]
Mutual Authentication	Security	12	TRUE [69]	TRUE [69]	TRUE [69]
Tunneling	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Reachability	Security	12	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif
Secrecy	Security	12	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif
Crypto-Agility	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Authentication Authority	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]

Continued on next page

Table 4 – *Continued from previous page*

Criteria	Category	Weight	WPA3 (EAP-TLS)	WPA3 (EAP-TTLS/PAP)	WPA3 (PEAP-MSCHAPv2)
Key Length	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Authentication Synchronization	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Authenticated Access	Security	12	TRUE [68], [69]	TRUE [68], [69]	TRUE [68], [69]
Authentication Logic	Security	12	TRUE (BAN logic)	TRUE (BAN logic)	TRUE (BAN logic)

Table 4: Authentication Protocol Evaluation 3

6.4 Authentication Protocol Evaluation 4

Criteria	Category	Weight	6LOWPAN	LonTalk	WPA2 (EAP-TLS)	WPA2 (EAP-TTLS)
Independence	Transport	1	TRUE [78]	TRUE [83]	TRUE [66], [67]	TRUE [66], [67]
Congestion Control	Transport	1	TRUE [78]	TRUE [83]	TRUE [66], [67]	TRUE [66], [67]
Message Matching	Transport	1	TRUE [78]	TRUE [131], [83]	TRUE [66], [67]	TRUE [66], [67]
Connection Types	Transport	1	FALSE	FALSE	FALSE	FALSE
IP	Transport	1	FALSE	FALSE	TRUE [66], [67]	TRUE [66], [67]
Data Types	Administration	2	TRUE [78]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Request and Response Management	Administration	2	TRUE [132]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Message Headers	Audit	3	TRUE [78]	TRUE [131], [83]	TRUE [66], [67]	TRUE [66], [67]
Accounting Management	Audit	3	TRUE [132]	FALSE	TRUE [66], [67]	TRUE [66], [67]
Scalability	Efficiency	6	TRUE [78]	TRUE [131]	TRUE [66], [67]	TRUE [66], [67]
Constrained	Efficiency	6	TRUE [78]	TRUE [131], [83]	FALSE	FALSE
Authentication Cluster	Security	12	TRUE [132]	FALSE	TRUE [66], [67]	TRUE [66], [67]
Error Detection and Correction	Security	12	TRUE [78]	FALSE	TRUE [66], [67]	TRUE [66], [67]
Encapsulation	Security	12	TRUE [78]	FALSE [131]	FALSE [66], [67]	FALSE [66], [67]
Encryption	Security	12	TRUE [78]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Certificates	Security	12	TRUE [78]	FALSE	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Integrity Check	Security	12	TRUE [132]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Mutual Authentication	Security	12	TRUE [78]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Tunneling	Security	12	TRUE [132]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Reachability	Security	12	TRUE ProVerif	FALSE ProVerif	TRUE ProVerif	TRUE Result
Secrecy	Security	12	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif	TRUE Result
Crypto-Agility	Security	12	TRUE [132]	FALSE	FALSE	FALSE
Authentication Authority	Security	12	TRUE [132]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]

Continued on next page

Table 5 – *Continued from previous page*

Criteria	Category	Weight	6LOWPAN	LonTalk	WPA2 (EAP-TLS)	WPA2 (EAP-TTLS)
Key Length	Security	12	TRUE [78]	FALSE	FALSE	FALSE
Authentication Synchronization	Security	12	TRUE [78]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Authenticated Access	Security	12	TRUE [78]	TRUE [131], [83]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Authentication Logic	Security	12	TRUE (BAN logic)	TRUE (BAN logic)	TRUE (BAN logic)	TRUE (BAN logic)

Table 5: Authentication Protocol Evaluation 4

6.5 Authentication Protocol Evaluation 5

Criteria	Category	Weight	WPA2 (PEAP-MSCHAPv2)	WPA&WPA2 (EAP-PSK)	WPA&WPA2-PSK
Independence	Transport	1	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Congestion Control	Transport	1	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Message Matching	Transport	1	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Connection Types	Transport	1	FALSE	FALSE	FALSE
IP	Transport	1	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Data Types	Administration	2	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Request and Response Management	Administration	2	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Message Headers	Audit	3	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Accounting Management	Audit	3	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Scalability	Efficiency	6	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Constrained	Efficiency	6	FALSE	FALSE	FALSE
Authentication Cluster	Security	12	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Error Detection and Correction	Security	12	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Encapsulation	Security	12	TRUE [66], [67]	TRUE [66], [67]	TRUE [66], [67]
Encryption	Security	12	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Certificates	Security	12	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Integrity Check	Security	12	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Mutual Authentication	Security	12	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Tunneling	Security	12	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Reachability	Security	12	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif
Secrecy	Security	12	TRUE ProVerif	TRUE ProVerif	TRUE ProVerif
Crypto-Agility	Security	12	FALSE	FALSE	FALSE
Authentication Authority	Security	12	TRUE [2], [66], [67]	TRUE [2], [66], [67]	TRUE [2], [66], [67]

Continued on next page

Table 6 – *Continued from previous page*

Criteria	Category	Weight	WPA2 (PEAP-MSCHAPv2)	WPA&WPA2 (EAP-PSK)	WPA&WPA2-PSK
Key Length	Security	12	FALSE	FALSE	FALSE
Authentication Synchronization	Security	12	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Authenticated Access	Security	12	TRUE [66], [67], [89]	TRUE [66], [67], [89]	TRUE [66], [67], [89]
Authentication Logic	Security	12	TRUE (BAN logic)	TRUE (BAN logic)	TRUE (BAN logic)

Table 6: Authentication Protocol Evaluation 5

6.6 Authentication Protocol Evaluation 6

Criteria	Category	Weight	KNX Secure	X10	INSTEON
Independence	Transport	1	TRUE [91]	FALSE	FALSE
Congestion Control	Transport	1	TRUE [91]	FALSE	TRUE [94]
Message Matching	Transport	1	TRUE [133]	TRUE [93]	TRUE [94]
Connection Types	Transport	1	TRUE	TRUE	TRUE
IP	Transport	1	TRUE [133]	FALSE	FALSE
Data Types	Administration	2	TRUE [133]	FALSE	TRUE [94]
Request and Response Management	Administration	2	TRUE [133]	TRUE [93]	TRUE [94]
Message Headers	Audit	3	TRUE [133]	FALSE	TRUE [94]
Accounting Management	Audit	3	FALSE	FALSE	TRUE [94]
Scalability	Efficiency	6	TRUE [91]	FALSE	TRUE [94]
Constrained	Efficiency	6	TRUE	TRUE [93]	TRUE [94]
Authentication Cluster	Security	12	FALSE	FALSE	FALSE
Error Detection and Correction	Security	12	TRUE [133]	TRUE [93]	TRUE [94]
Encapsulation	Security	12	TRUE [134]	FALSE	FALSE
Encryption	Security	12	TRUE [135]	FALSE	FALSE
Certificates	Security	12	TRUE [134]	FALSE	FALSE
Integrity Check	Security	12	TRUE [134]	FALSE	TRUE [94]
Mutual Authentication	Security	12	TRUE [135]	FALSE	FALSE
Tunneling	Security	12	TRUE [133]	FALSE	FALSE
Reachability	Security	12	TRUE ProVerif	FALSE	FALSE
Secrecy	Security	12	TRUE ProVerif	FALSE	FALSE
Crypto-Agility	Security	12	FALSE	FALSE	FALSE
Authentication Authority	Security	12	TRUE [134]	FALSE	FALSE

Continued on next page

Table 7 – *Continued from previous page*

Criteria	Category	Weight	KNX Secure	X10	INSTEON
Key Length	Security	12	FALSE	FALSE	FALSE
Authentication Synchronization	Security	12	TRUE [135]	FALSE	FALSE
Authenticated Access	Security	12	TRUE	FALSE	TRUE
Authentication Logic	Security	12	TRUE (BAN logic)	FALSE	FALSE

Table 7: Authentication Protocol Evaluation 6

6.7 Authentication Protocol Evaluation 7

Criteria	Category	Weight	ANT+	Z-Wave	G.hn	UPB
Independence	Transport	1	FALSE	TRUE [6]	TRUE [82]	FALSE
Congestion Control	Transport	1	FALSE	TRUE [6]	TRUE [82]	FALSE
Message Matching	Transport	1	TRUE	[6], [136]	TRUE [82]	TRUE [90]
Connection Types	Transport	1	FALSE	FALSE	TRUE [82]	FALSE
IP	Transport	1	FALSE	TRUE [6], [136]	TRUE [82]	FALSE
Data Types	Administration	2	TRUE [137]	TRUE [6], [136]	FALSE [82]	FALSE
Request and Response Management	Administration	2	TRUE [137]	TRUE [138]	TRUE [82], [80]	TRUE [90]
Message Headers	Audit	3	TRUE [54]	TRUE [6], [136]	TRUE [82]	FALSE
Accounting Management	Audit	3	FALSE	FALSE	FALSE	FALSE
Scalability	Efficiency	6	TRUE [137]	TRUE [61]	FALSE	FALSE
Constrained	Efficiency	6	TRUE [54]	TRUE [6], [136]	TRUE [82]	TRUE [90]
Authentication Cluster	Security	12	FALSE	FALSE	FALSE	FALSE
Error Detection and Correction	Security	12	TRUE [54]	TRUE [6], [136]	TRUE [82]	TRUE [90]
Encapsulation	Security	12	FALSE	TRUE [138]	FALSE	FALSE
Encryption	Security	12	FALSE	[6], [136]	TRUE [82], [80]	FALSE
Certificates	Security	12	FALSE	TRUE [6], [136]	FALSE	FALSE
Integrity Check	Security	12	FALSE	TRUE [6], [136]	TRUE [82]	FALSE
Mutual Authentication	Security	12	TRUE [137]	TRUE [6], [136]	TRUE [82], [80]	FALSE
Tunneling	Security	12	FALSE	TRUE [6], [136]	FALSE	FALSE
Reachability	Security	12	FALSE	TRUE ProVerif	TRUE ProVerif	FALSE
Secrecy	Security	12	FALSE	TRUE ProVerif	TRUE ProVerif	FALSE
Crypto-Agility	Security	12	FALSE	FALSE	FALSE	FALSE
Authentication Authority	Security	12	TRUE [137]	TRUE [6], [136]	TRUE [82], [80]	TRUE

Continued on next page

Table 8 – *Continued from previous page*

Criteria	Category	Weight	ANT+	Z-Wave	G.hn	UPB
Key Length	Security	12	FALSE	TRUE [6], [136]	TRUE [82], [80]	FALSE
Authentication Synchronization	Security	12	FALSE	TRUE [6], [136]	TRUE [82], [80]	FALSE
Authenticated Access	Security	12	TRUE [137]	TRUE [6], [136]	TRUE [82], [80]	TRUE
Authentication Logic	Security	12	FALSE	TRUE (BAN logic)	TRUE (BAN logic)	FALSE

Table 8: Authentication Protocol Evaluation 7

6.8 Authentication Protocol Evaluation 8

Criteria	Category	Weight	IrDA	Dash7	UWB	BLE
Independence	Transport	1	FALSE	FALSE	FALSE	FALSE
Congestion Control	Transport	1	TRUE [3], [37]	TRUE [45]	TRUE [70]	FALSE
Message Matching	Transport	1	TRUE	TRUE [45]	TRUE [70]	TRUE
Connection Types	Transport	1	FALSE	FALSE	FALSE	FALSE
IP	Transport	1	FALSE	TRUE [121]	FALSE [70]	FALSE
Data Types	Administration	2	FALSE	TRUE	TRUE [70]	FALSE
Request and Response Management	Administration	2	TRUE [37]	TRUE [45]	TRUE [70]	TRUE [139]
Message Headers	Audit	3	FALSE	TRUE [45]	TRUE [70]	TRUE [140]
Accounting Management	Audit	3	FALSE	FALSE	FALSE	FALSE
Scalability	Efficiency	6	FALSE	FALSE	TRUE [70]	FALSE
Constrained	Efficiency	6	TRUE [37]	TRUE [141]	TRUE [70], [72]	TRUE [142]
Authentication Cluster	Security	12	FALSE	FALSE	FALSE	FALSE
Error Detection and Correction	Security	12	TRUE [37]	TRUE [45]	TRUE [70]	TRUE [51]
Encapsulation	Security	12	FALSE	TRUE [45]	FALSE	FALSE
Encryption	Security	12	FALSE	TRUE [143]	TRUE [70]	TRUE [51]
Certificates	Security	12	FALSE	FALSE	FALSE	FALSE
Integrity Check	Security	12	TRUE [37]	TRUE [45]	TRUE [70]	TRUE [51]
Mutual Authentication	Security	12	FALSE	TRUE [143]	TRUE [70]	TRUE [51]
Tunneling	Security	12	FALSE	FALSE	FALSE	FALSE
Reachability	Security	12	FALSE	FALSE	FALSE ProVerif	TRUE Proverif
Secrecy	Security	12	FALSE	FALSE	TRUE ProVerif	TRUE Proverif
Crypto-Agility	Security	12	FALSE	FALSE	FALSE	FALSE
Authentication Authority	Security	12	FALSE	FALSE	FALSE	TRUE [51]

Continued on next page

Table 9 – *Continued from previous page*

Criteria	Category	Weight	IrDA	Dash7	UWB	BLE
Key Length	Security	12	FALSE	TRUE [143]	TRUE [70]	FALSE [51]
Authentication Synchronization	Security	12	FALSE	TRUE [143]	TRUE [70]	TRUE [51]
Authenticated Access	Security	12	FALSE	FALSE	TRUE [70]	TRUE [51]
Authentication Logic	Security	12	FALSE	TRUE (BAN logic)	TRUE (BAN logic)	TRUE (BAN logic)

Table 9: Authentication Protocol Evaluation 8

6.9 Authentication Protocol Evaluation 9

Criteria	Category	Weight	Wireless HART	WiMAX v1	WiMAX v2
Independence	Transport	1	TRUE [98]	FALSE	FALSE
Congestion Control	Transport	1	TRUE [97]	TRUE [74], [73]	TRUE [74], [73]
Message Matching	Transport	1	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Connection Types	Transport	1	TRUE	FALSE	FALSE
IP	Transport	1	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Data Types	Administration	2	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Request and Response Management	Administration	2	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Message Headers	Audit	3	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Accounting Management	Audit	3	FALSE	FALSE	FALSE
Scalability	Efficiency	6	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Constrained	Efficiency	6	[98]	TRUE [74], [73]	TRUE [74], [73]
Authentication Cluster	Security	12	FALSE	FALSE	FALSE
Error Detection and Correction	Security	12	TRUE [97]	TRUE [74], [73]	TRUE [74], [73]
Encapsulation	Security	12	FALSE	FALSE	FALSE
Encryption	Security	12	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Certificates	Security	12	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Integrity Check	Security	12	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Mutual Authentication	Security	12	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Tunneling	Security	12	FALSE	FALSE	FALSE
Reachability	Security	12	FALSE	FALSE ProVerif	TRUE ProVerif
Secrecy	Security	12	FALSE	TRUE ProVerif	ProVerif
Crypto-Agility	Security	12	TRUE [74], [73]	TRUE [74], [73]	TRUE [74], [73]
Authentication Authority	Security	12	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]

Continued on next page

Table 10 – *Continued from previous page*

Criteria	Category	Weight	Wireless HART	WiMAX PKMv1	WiMAX PKMv2
Key Length	Security	12	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Authentication Synchronization	Security	12	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Authenticated Access	Security	12	TRUE [98]	TRUE [74], [73]	TRUE [74], [73]
Authentication Logic	Security	12	TRUE (BAN logic)	TRUE (BAN logic)	TRUE (BAN logic)

Table 10: Authentication Protocol Evaluation 9

6.10 Scoring Result

Protocol	Total Score
6LOWPAN	0.885714286
WPA3 (EAP-TLS)	0.816326531
WPA3 (EAP-TTLS/PAP)	0.816326531
WPA3 (PEAP-MSCHAPv2)	0.816326531
Zigbee SEP	0.771428571
KNX	0.734693878
Z-Wave	0.730612245
WPA2 (EAP-TLS)	0.718367347
WPA2 (EAP-TTLS/PAP)	0.718367347
WPA2 (PEAP-MSCHAPv2)	0.718367347
4G	0.702040816
3G	0.697959184
WiMAX PKMv2	0.67755102
Thread	0.628571429
WiMAX PKMv1	0.628571429
WPA (EAP-PSK)	0.620408163
WPA2 (PSK)	0.620408163
G.HN	0.604081633
Wireless HART	0.587755102
Lontalk	0.579591837
EnOcean SP3 Mutual	0.546938776
EnOcean SP3 Unilateral	0.546938776
Bluetooth Low Energy (BLE)	0.53877551
UWB	0.514285714
2G	0.453061224
Dash7	0.444897959
ANT+	0.27755102
Insteon	0.248979592
UPB	0.183673469
IrDA	0.13877551
X10	0.089795918

Table 11: Authentication Protocol Scoring Results

7 Conclusion

According to this method most secure protocol is 6LOWPAN, next would be WPA3 technology based authentication protocols. The same statement is true as well to the most security criterion achieved protocol. X10 received the lowest score, following IrDA and with Insteon and UPB tied. Powerline communication protocols were expected to be at the bottom of the table as well. If we would take efficiency to account, then most secure efficient, meaning both constrained and scalability criterion are True, is ZigBee SEP and next, Z-Wave tied with KNX. Which have their own smart home technology as well. Unfortunately this method showed that some protocols, which are widely used, are not fit for use on the public channel, for example Insteon. Insteon uses IP network as well and relies on physical security only, this protocol would not be fit for smart grid, but while it is actually designed for smart homes, then it is still the users choice, which technology he or she is going to use.

Furthermore, it is understandable that not all smart home devices do not to be authenticated, for example lighting system. It is just not feasible, but to join the actual network, where communication takes place, that is something what must be restricted. One should not be able to control smart home or smart grid devices with just plugging into the power line network or local are network. Nevertheless, smart home and smart grid protocols have a long way to go, but as it seems, the higher the power consumption the higher the security, which can also be expected. So far, there has not been technological breakthrough in computational strength with low power processing units, but while smart grids being developed and smart homes becoming more common than ever and a network containing of smart homes and more do not seem so unfeasible anymore, then this is seems like a way to go.

According to these results, it is clear that depreciation of X10 should be considered, even IrDA at least for smart home and smart home networks. Powerline communication has a long way to go, comparing it to radio frequency or cellular based communication.

Truly seems like the title of the most suitable communication standard for smart grid residential network domain by the U.S. National Institute for Standards and Technology, which was give for ZigBee, is something that is deserved. This also verifies that with this method authentication protocols can be evaluated and get meaningful results.

8 Future Uses

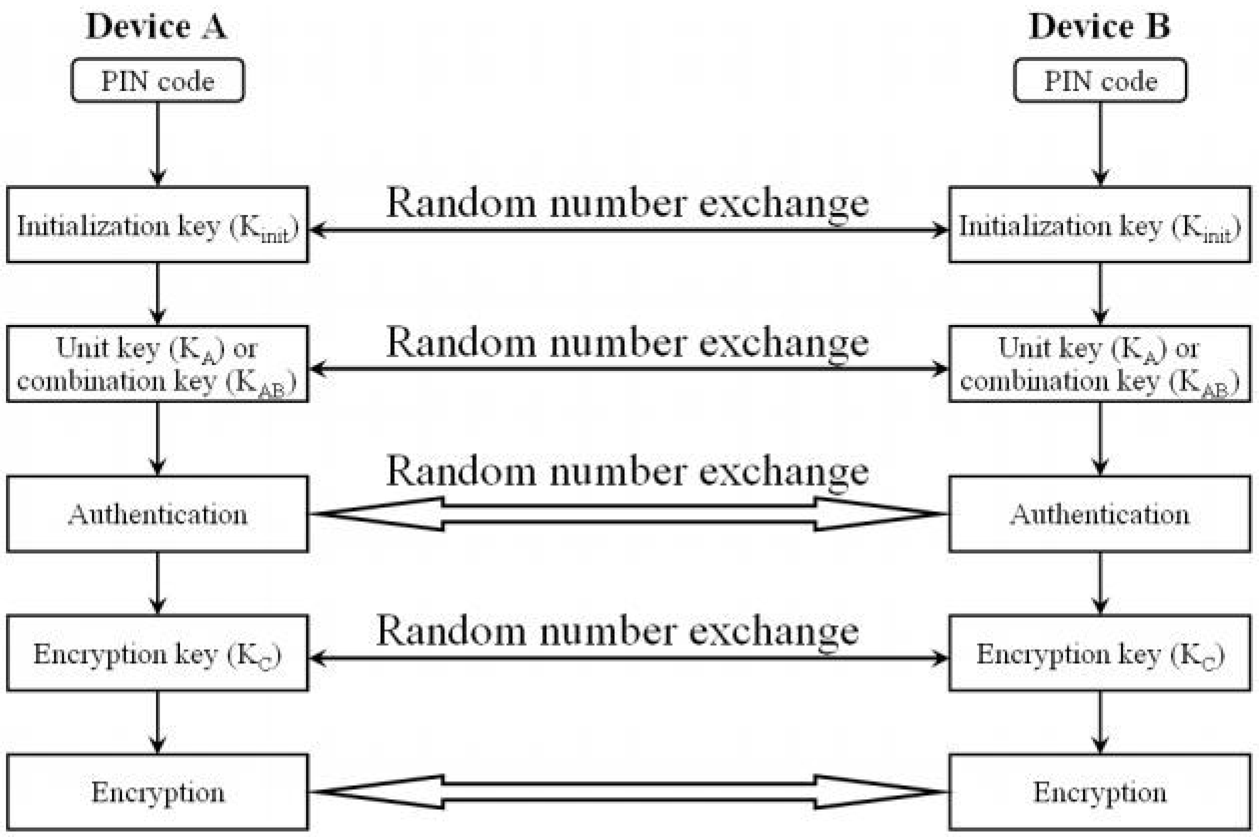
This method can be improved a lot, for example evaluate the computational strength of the used encryption algorithms and elaborate key strength criteria. Also one could evaluate the computational power of these devices which are used by one's technology. Even going deeper into verification of authentication, using involing more tools and criterion.
One might even be able to develop perfect protocol according to this method. Unfortunately still some external measures have to be taken into account. Sound and secure communication protocols cannot be developed over night, just like evaluation methods. They must be constantly improved and be used, so that they could grow into something, that would emerge into a standard.

Appendices

Appendix 1 – Bluetooth Authentication

Bluetooth uses for pairing Elliptic Curve Diffie Hellman based key agreement, which is conducted as follows [3]: Two Bluetooth devices begin communication with the same PIN (Personal Identification Number) code that is used for generating 128-bit random numbers. Each master and slave pair can have a different PIN code for the devices.
An initialization key is generated when Bluetooth devices meet for the first time and is used for securing the generation of other more secure 128-bit keys. An initialization key is derived from an unencrypted 128-bit random number IN_RAND, an L-byte ($1 \leq L \leq 16$) PIN code, and a BD_ADDR. If one device has a fixed PIN code, the BD_ADDR of the other device is used. If both devices can support a variable PIN code, the BD_ADDR of the device that received IN_RAND is used.
The initialization key is used for encrypting a 128-bit random number LK_RAND, which is exchanged when the next link or a key is generated. A combination key is always dependent on two devices and therefore derived from information of both devices (BD_ADDRA, LK_RANDA, BD_ADDRB, LK_RANDB). It is used for challenge-response authentication in which an asker's knowledge of a secret link key is checked.
During each authentication, a new 128-bit unencrypted random number AU_RAND is exchanged. The asker returns a 32-bit result (SRES, Signed Response) to the verifier. The verifier also calculates the same SRES value and compares it to the received SRES. If the SRES values match, the authentication is completed successfully and a 96-bit result (ACO, Authenticated Ciphering Offset) is computed in both devices.
An ACO, a link key and an unencrypted 128-bit random number EN_RAND are used for generating an encryption key, which is one input to the keystream generator and makes symmetric encryption possible. Other inputs to the keystream generator are master's BD_ADDR and 26 bits of the master's real-time clock. Application layer key exchange and encryption methods can also be used to secure communication on top of the existing Bluetooth security measures. Bluetooth security has remained almost unchanged since the first Bluetooth 1.0 specification was released 1999. [3] Diffie-Hellman key exchange has been explained in Appendix 3. Message exchange for simplified BLE authentication sequence is show in Figure 2:

Figure 2: Bluetooth authentication dialogue [3]



ProVerif Verification

```
set traceDisplay = long.

query attacker(DecryptedText4).
query attacker(DecryptedText5).
query attacker(LongTermKey).
query attacker(LongTermKeyBitstring).
query x:bitstring; event(Decryption4(x)).
query x:bitstring; event(Decryption5(x)).
query x:bitstring; event(Decryption4(x)) ==> event(Decryption5(x)).
query x:bitstring; inj-event(Decryption4(x)) ==> inj-event(Decryption5(x))
    .

type pin.
type key.
type nonce.
type realkey.
free DeviceAPin:pin[private].
free DeviceBPin:pin[private].
free c: channel.
free Hello:bitstring [private].
free HelloACK, TESTACK:bitstring [private].
free RAND_NR:bitstring[private].
free INITKEY1:key[private].
free AU_RAND_B:bitstring[private].
free INITKEY2:key[private].
free AU_RAND_A, TEST:bitstring[private].
free DecryptedText1, DecryptedText2, DecryptedText3,DecryptedText4,
   DecryptedText5, LongTermKeyBitstring:bitstring[private].
free LongTermKey:realkey[private].

fun InitKeyGen(bitstring, bitstring, pin): key.
fun SRESGen(key, bitstring, bitstring): bitstring.
fun NonceToRealKey(bitstring):realkey [typeConverter].
reduc forall x: key, y: bitstring, z:bitstring ; SRESDeGen(SRESGen(x, y, z
   ), y, z) = x.

fun Encryption(bitstring, key): nonce.
reduc forall x: bitstring, y: key; Decryption(Encryption(x, y), y) = x.

fun RealEncryption(bitstring, realkey): nonce.
```

```
reduc forall x: bitstring, y: realkey; RealDecryption(RealEncryption(x, y)
   , y) = x.

event RandomNumberGeneration(bitstring).
event RandomNumberGeneration2(bitstring).
event InitKeyGeneration1(key).
event InitKeyGeneration2(key).
event SRESGeneration(bitstring).
event SRESMATCH(bitstring).
event Decryption1(bitstring).
event Decryption2(bitstring).
event Decryption3(bitstring).
event Decryption4(bitstring).
event Decryption5(bitstring).
event LongTermKeyGeneration(realkey).

let DeviceB1 =
        new RAND_NR2:bitstring;
        let AU_RAND_B = RAND_NR2 in
                event RandomNumberGeneration2(AU_RAND_B);
        out(c, AU_RAND_B).

let DeviceA1 =
        in(c, AU_RAND_B:bitstring);
        new RAND_NR:bitstring;
        let AU_RAND_A = RAND_NR in
                event RandomNumberGeneration(AU_RAND_A);
        out(c, AU_RAND_A).

let DeviceB2 =
        in(c, AU_RAND_A:bitstring);
        let INITKEY2 = InitKeyGen(AU_RAND_B, AU_RAND_A, DeviceBPin) in
                event InitKeyGeneration2(INITKEY2);
        let SRESB = SRESGen(INITKEY2,AU_RAND_B,AU_RAND_A) in
                event SRESGeneration(SRESB);
        out(c, SRESB).

let DeviceA2 =
        in(c, SRESB:bitstring);
        let INITKEY1 = InitKeyGen(AU_RAND_B, AU_RAND_A, DeviceAPin) in
                event InitKeyGeneration1(INITKEY1);
        if SRESB = SRESGen(INITKEY1,AU_RAND_B,AU_RAND_A) then
        (
                event SRESMATCH(SRESB);
                out(c, Encryption(Hello, INITKEY1))
        )
        else
        (
                0
        ).

let DeviceB3 =
        in(c, EncryptedText1:nonce);
        let DecryptedText1 = Decryption(EncryptedText1, INITKEY2) in
                event Decryption1(DecryptedText1);
        out(c, Encryption(HelloACK, INITKEY2)).

let DeviceA3 =
        in(c, EncryptedText2:nonce);
        let DecryptedText2 = Decryption(EncryptedText2, INITKEY1) in
                event Decryption2(DecryptedText2);
        new LongTermKey:realkey;
        new LongTermKeyBitstring:bitstring;
        let LongTermKey = LongTermKey in
                event LongTermKeyGeneration(LongTermKey);
        out(c, Encryption(LongTermKeyBitstring, INITKEY1)).

        let DeviceB4 =
                in(c, EncryptedText3:nonce);
                let DecryptedText3 = Decryption(EncryptedText3, INITKEY2)
                   in
                        event Decryption3(DecryptedText3);
                out(c, RealEncryption(TEST, NonceToRealKey(DecryptedText3)
                   )).

        let DeviceA4 =
                in(c, EncryptedText4:nonce);
                let DecryptedText4 = RealDecryption(EncryptedText4,
                   LongTermKey) in
                        event Decryption4(DecryptedText4);
                out(c, RealEncryption(TESTACK, LongTermKey)).

        let DeviceB5=
        in(c, EncryptedText5:nonce);
        let DecryptedText5 = RealDecryption(EncryptedText5, NonceToRealKey
           (DecryptedText3)) in
               event Decryption5(DecryptedText5).

process (!DeviceA1|!DeviceB1|!DeviceA2|!DeviceB2|!DeviceA3|!DeviceB4|!
   DeviceA4|!DeviceB5)
```

Proverif Result

```
Process:
(
    {1}!
    {2}in(c, AU_RAND_B_35: bitstring);
    {3}new RAND_NR_36: bitstring;
    {4}let AU_RAND_A_37: bitstring = RAND_NR_36 in
    {5}event RandomNumberGeneration(AU_RAND_A_37);
    {6}out(c, AU_RAND_A_37)
) | (
    {7}!
    {8}new RAND_NR2: bitstring;
    {9}let AU_RAND_B_38: bitstring = RAND_NR2 in
    {10}event RandomNumberGeneration2(AU_RAND_B_38);
    {11}out(c, AU_RAND_B_38)
) | (
    {12}!
    {13}in(c, SRESB: bitstring);
    {14}let INITKEY1_39: key = InitKeyGen(AU_RAND_B,AU_RAND_A,DeviceAPin)
       in
    {15}event InitKeyGeneration1(INITKEY1_39);
    {16}if (SRESB = SRESGen(INITKEY1_39,AU_RAND_B,AU_RAND_A)) then
    {17}event SRESMATCH(SRESB);
    {18}out(c, Encryption(Hello,INITKEY1_39))
) | (
    {19}!
    {20}in(c, AU_RAND_A_40: bitstring);
    {21}let INITKEY2_41: key = InitKeyGen(AU_RAND_B,AU_RAND_A_40,
       DeviceBPin) in
    {22}event InitKeyGeneration2(INITKEY2_41);
    {23}let SRESB_42: bitstring = SRESGen(INITKEY2_41,AU_RAND_B,
       AU_RAND_A_40) in
    {24}event SRESGeneration(SRESB_42);
    {25}out(c, SRESB_42)
) | (
    {26}!
    {27}in(c, EncryptedText2: nonce);
    {28}let DecryptedText2_43: bitstring = Decryption(EncryptedText2,
       INITKEY1) in
    {29}event Decryption2(DecryptedText2_43);
    {30}new LongTermKey_44: realkey;
    {31}new LongTermKeyBitstring_45: bitstring;
    {32}let LongTermKey_46: realkey = LongTermKey_44 in
    {33}event LongTermKeyGeneration(LongTermKey_46);
    {34}out(c, Encryption(LongTermKeyBitstring_45,INITKEY1))
) | (
    {35}!
    {36}in(c, EncryptedText3: nonce);
    {37}let DecryptedText3_47: bitstring = Decryption(EncryptedText3,
       INITKEY2) in
    {38}event Decryption3(DecryptedText3_47);
    {39}out(c, RealEncryption(TEST,DecryptedText3_47))
) | (
    {40}!
    {41}in(c, EncryptedText4: nonce);
    {42}let DecryptedText4_48: bitstring = RealDecryption(EncryptedText4,
       LongTermKey) in
    {43}event Decryption4(DecryptedText4_48);
    {44}out(c, RealEncryption(TESTACK,LongTermKey))
) | (
    {45}!
    {46}in(c, EncryptedText5: nonce);
    {47}let DecryptedText5_49: bitstring = RealDecryption(EncryptedText5,
       DecryptedText3) in
    {48}event Decryption5(DecryptedText5_49)
)

-- Query not attacker(DecryptedText4[])
nounif attacker(Encryption(DecryptedText2_337,INITKEY1[]))/-5000
Completing...
Starting query not attacker(DecryptedText4[])
RESULT not attacker(DecryptedText4[]) is true.
-- Query not attacker(DecryptedText5[])
nounif attacker(Encryption(DecryptedText2_732,INITKEY1[]))/-5000
Completing...
Starting query not attacker(DecryptedText5[])
RESULT not attacker(DecryptedText5[]) is true.
-- Query not attacker(LongTermKey[])
nounif attacker(Encryption(DecryptedText2_1131,INITKEY1[]))/-5000
Completing...
Starting query not attacker(LongTermKey[])
RESULT not attacker(LongTermKey[]) is true.
-- Query not attacker(LongTermKeyBitstring[])
```

```
nounif attacker(Encryption(DecryptedText2_1530,INITKEY1[]))/-5000
Completing...
Starting query not attacker(LongTermKeyBitstring[])
RESULT not attacker(LongTermKeyBitstring[]) is true.
-- Query not event(Decryption4(x_50))
nounif attacker(Encryption(DecryptedText2_1929,INITKEY1[]))/-5000
Completing...
Starting query not event(Decryption4(x_50))
RESULT not event(Decryption4(x_50)) is true.
-- Query not event(Decryption5(x_51))
nounif attacker(Encryption(DecryptedText2_2333,INITKEY1[]))/-5000
Completing...
Starting query not event(Decryption5(x_51))
RESULT not event(Decryption5(x_51)) is true.
-- Query event(Decryption4(x_52)) ==> event(Decryption5(x_52))
nounif attacker(Encryption(DecryptedText2_2737,INITKEY1[]))/-5000
Completing...
Starting query event(Decryption4(x_52)) ==> event(Decryption5(x_52))
RESULT event(Decryption4(x_52)) ==> event(Decryption5(x_52)) is true.
-- Query inj-event(Decryption4(x_53)) ==> inj-event(Decryption5(x_53))
nounif attacker(Encryption(DecryptedText2_3141,INITKEY1[]))/-5000
Completing...
Starting query inj-event(Decryption4(x_53)) ==> inj-event(Decryption5(x_53
    ))
RESULT inj-event(Decryption4(x_53)) ==> inj-event(Decryption5(x_53)) is
    true.
```

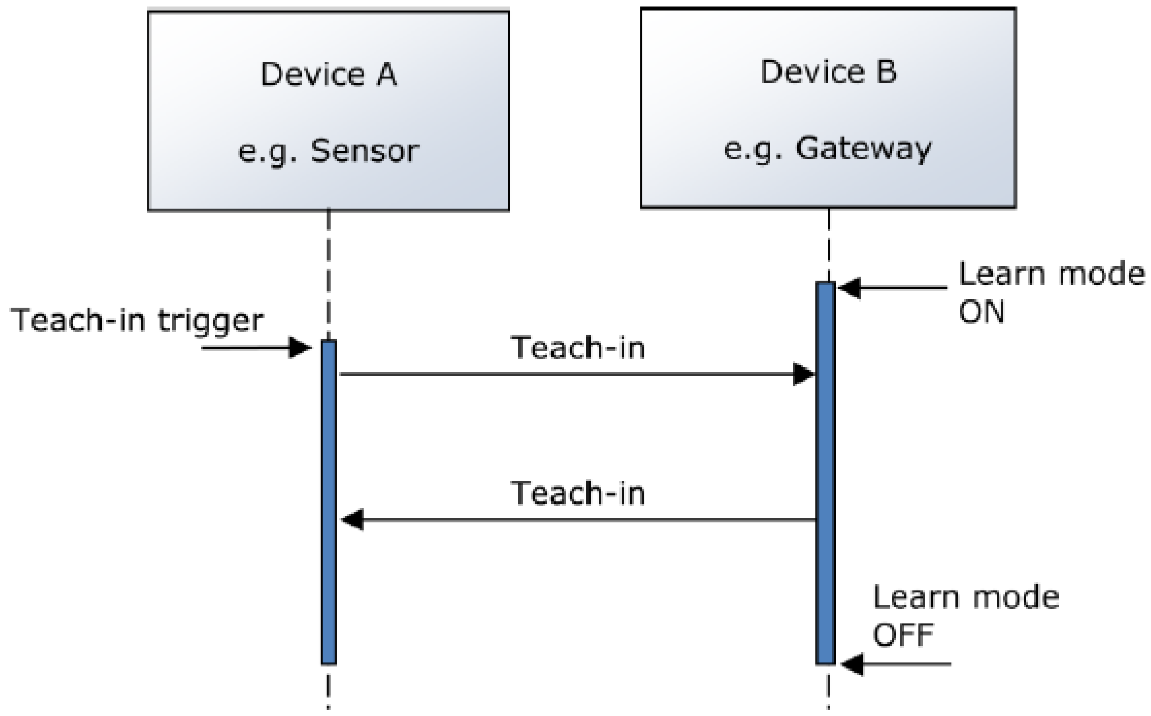
Appendix 2 – EnOcean Authentication

To configure the information needed for the secure communication in operation mode a teach-in procedure mode must be executed. Within the teach-in procedure following information are transmitted to one another : the encryption method, key, rolling code, rolling code size and CMAC size that will be used during the operation mode. The teach-in procedure can be set up to be an unidirectional or bidirectional process. [4]
In the case of unidirectional security teach-in Device B does not send a teach-in message. Firstly, the Device B must be set in its learn mode to accept the teach-in messages from Device A.The Device A sends the security teach-in message whenever its specific trigger is activated.After reception of the teach-in message the Device B stores the security parameters of Device A: these parameters include the Device's A private key, KEYs, current RLC, RLCs, RLCs size and CMACs size and way of encrypting information. The KEYs and RLCs can be sent encrypted by the sender using the so called pre-shared key, PSKs. [4]
If the process is bidirectional the Device B, a gateway, for instance, answers back with a security teach-in message. This teach-in message contains as receiver-ID the ID of the Device A. If the Device B encrypts its teach-in message it will make use of the same PSKs key of the Device A. In the second security teach-in depicted in the picture the Device B informs the Device A of its own KEYg and RLCg and CMACg. The format of the teach-in messages sent by Device A and Device B are the same. The teach-in delivered by Device B must occur in worst case 500ms after the reception of the teach-in sent by Device A. The Device's A time-out for the reception of a teach-in is 750ms. Before Device A sends the security teach-in message the receiver is put into teach-in mode – active listening for teach-in messages. The teach-in method is limited typically to 30 seconds. After this time-out the module leaves its teach-in mode, and returns typically to its operation mode. Teach-in messages are not accepted until the next activation of the teach-in mode. [4]
EnOcean SP3 Teach-in message exchange is shown in Figure 3. Methods for the teach-in for execution

1. Over wireless from the transmitter to the receiver
2. Over serial interface to the receiver through a third party

Execution over serial interface or other methods are not part of this specification and are rather application / use case specific. The execution of the teach-in process via wireless

Figure 3: EnOcean Teach-In Process [4]



leads to two possibilities:

1. Teach-in message is sent in plain text (no encryption in the information is performed). This means that any listener can eavesdrop the information.

2. Parts of the teach-in message are encrypted. For the encryption a pre-shared key is used. Encrypted are the RLC and KEY. Message structure is listed below. Details about this execution can be found in chapter

This secure teach-in message only transfers the security specific data. To enable profile interpretation a profile-teach-in message (EEP or GP) has to be transmitted after the secure teach-in. This profile teach-in is conducted already secured (encrypted) using the decrypted key that the secure teach-in transmitted. With the information contained in the teach-in SLF byte the receiver is informed about the details of the secure messages in operation mode: what fields, how long they are, and what are the applied security algorithms. The pre-shared key of the sender module must have been communicated to the receiver (a gateway) via serial interface in advance. The pre-shared key is typically written on a sticker on the sender module. The pre-shared key is not transmitted through the EnOcean air interface. If pre-shared key is used RLC and KEY will be encrypted using the VAES encryption. [4]

Authentication based on CMAC Authentication can be:

1. Unilateral – only one of the communication partners is authenticated and his outgoing communication is protected against replay attacks – one Nonce is exchanged.
2. Mutual – both communication partners are authenticated and both communication ways are protected against replay attacks – two Nonce are exchanged

For CMAC computing the EnOcean security concept uses the Payload of telegrams. The Nonce is used during CMAC computing too and so ensures that the Nonce is connected with the exchanged message and its data content. So becomes the data content also valid for limited time. EnOcean Security concept uses the VAES for data encryption. The Nonce can be also used for the initialization vector for the VAES process. This way a random element is added to the VAES process. This is required if Nonce is a random number and no RLC is used. [4] The Nonce can be:

1. Used during CMAC counting
2. Used as initialization vector for VAES (encryption/decryption) counting

Nonce represents the challenge and has to be therefore exchanged between the communication partners via air interface. The Challenge and Response does not have to encrypt and can be transferred plaintext. The CMAC algorithm represents the securing element. During data communication following constrains are applied:

1. Bidirectional communication can be only executed after mutual authentication. Both parties can trigger the authentication.
2. Unidirectional communication is unilateral authenticated. The emitter of the data flow is authenticated and only the emitter can trigger the communication. The challenge is provided by the consumer of the data.

For Nonce one can use:

1. Random number 32 bit – here is critical that the generator process is not predictable, does not repeat same sequences and is equally spread on the defined range
2. Simple incrementing, non-repeating sequence 32-bit number

Figure 4: EnOcean mutual authentication [4]

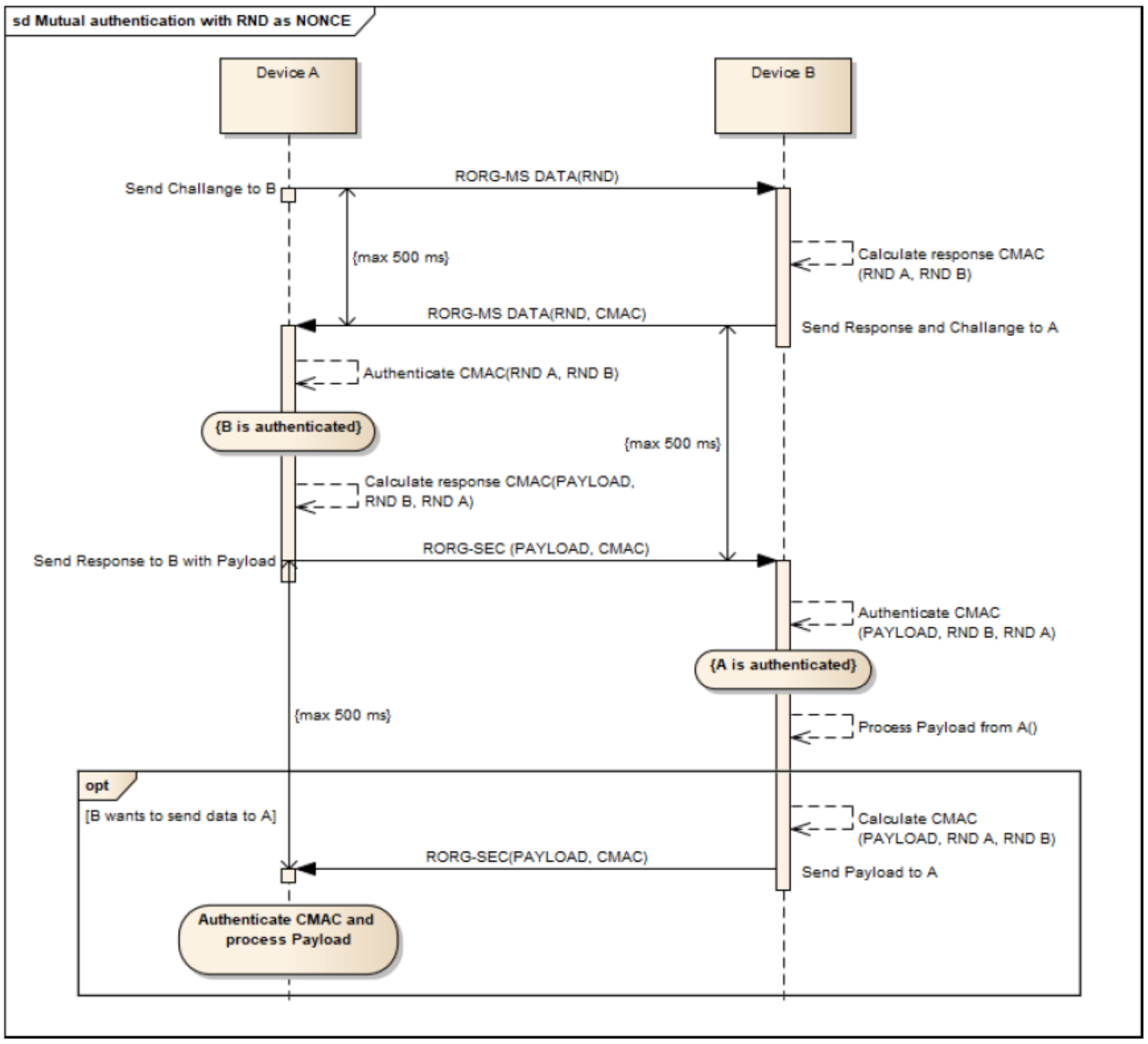
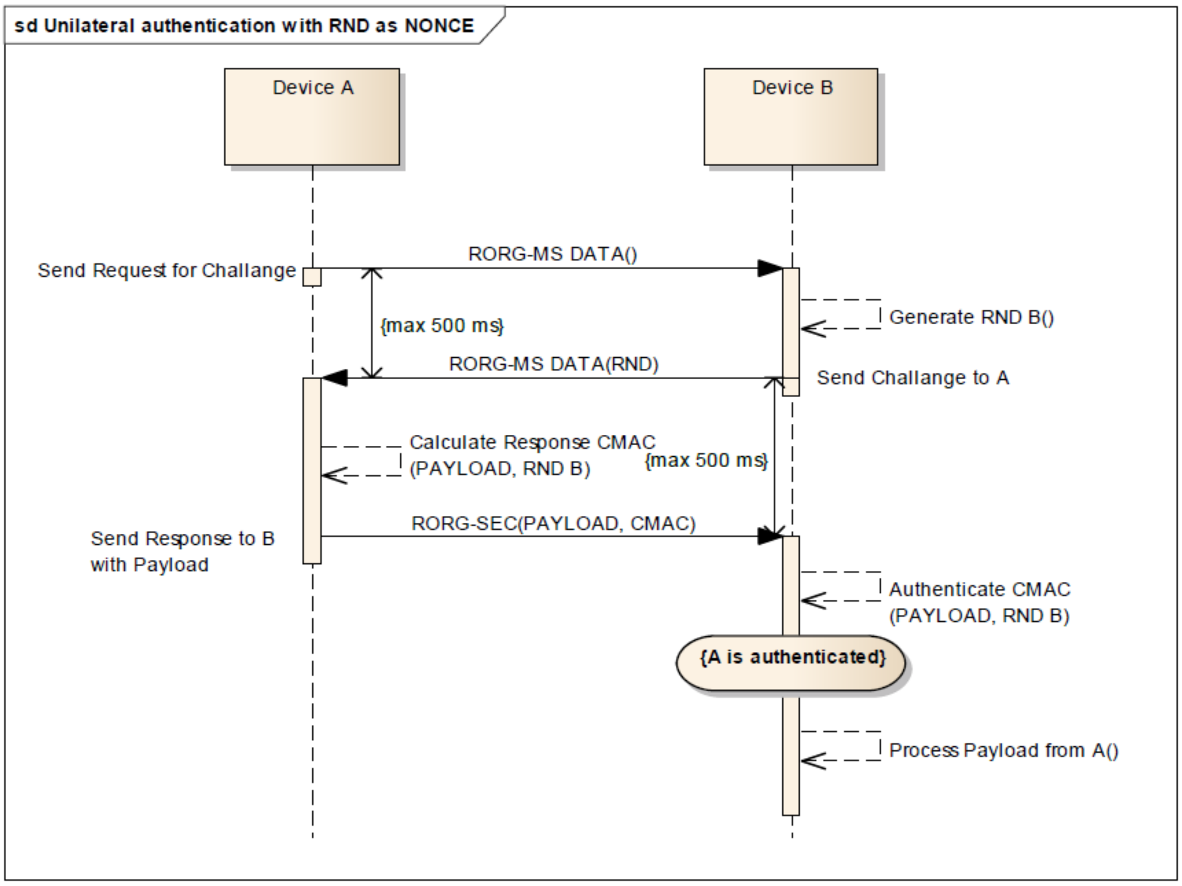

sd Mutual authentication with RND as NONCE
Device A
Device B
RORG-MS DATA(RND)
Send Challange to B
{max 500 ms}
Calculate response CMAC (RND A, RND B)
RORG-MS DATA(RND, CMAC)
Send Response and Challange to A
Authenticate CMAC(RND A, RND B)
{B is authenticated}
{max 500 ms}
Calculate response CMAC(PAYLOAD, RND B, RND A)
RORG-SEC (PAYLOAD, CMAC)
Send Response to B with Payload
Authenticate CMAC (PAYLOAD, RND B, RND A)
{A is authenticated}
{max 500 ms}
Process Payload from A()
opt
[B wants to send data to A]
Calculate CMAC (PAYLOAD, RND A, RND B)
RORG-SEC(PAYLOAD, CMAC)
Send Payload to A
Authenticate CMAC and process Payload

Figure 5: EnOcean Unilateral authentication [4]


sd Unilateral authentication with RND as NONCE
Device A
Device B
RORG-MS DATA()
Send Request for Challange
Generate RND B()
{max 500 ms}
RORG-MS DATA(RND)
Send Challange to A
Calculate Response CMAC (PAYLOAD, RND B)
{max 500 ms}
RORG-SEC(PAYLOAD, CMAC)
Send Response to B with Payload
Authenticate CMAC (PAYLOAD, RND B)
{A is authenticated}
Process Payload from A()

EnOcean SP3 mutual authentication message exchange is shown in Figure 4 and unilateral authentication Figure 5.

If the message gets lost the process is considered as failed. Repeated transmissions of request for challenge or challenge messages between identical communication partners shall restart the authentication process and cancel any previous ongoing validation. The PSK code (16 bytes) comes together with an extra byte, CRC checksum, which is used to verify that the installer writes correctly the 16-byte PSK code into the Device B. The checksum uses a CRC8 algorithm. Data can be encrypted using the standard high-security AES128 algorithm with cipher-block chaining (CBC). Constant data will result in constant encrypted information. [4]
CMAC authentication is showin in Appendix 18

Mutual Authentication ProVerif Verification

```
set traceDisplay = long.

query attacker(SecureKey).
query attacker(SecureKeyBitstring).

query attacker(DecryptedData4).
query attacker(DecryptedData5).

query x:bitstring; event(DataDecryption4(x)).
query x:bitstring; event(DataDecryption5(x)).
query x:bitstring; event(DataDecryption4(x)) ==> event(DataDecryption5(x))
    .
query x:bitstring; inj-event(DataDecryption4(x)) ==> inj-event(
    DataDecryption5(x)).

free c:channel.
type nonce.
type key.
type seckey.

free N:nonce [private].

fun CMAC(bitstring, bitstring, bitstring, nonce, key):nonce.
reduc forall x: bitstring, y: bitstring, v: bitstring, z: nonce, c:key;
    DeCMAC(CMAC(x,y,v,z,c),c) = x.

free CMACData1: nonce [private].
free CMACData2: nonce [private].

free Data:bitstring [private].
free RND_NumberA:bitstring [private].
free RND_NumberB:bitstring [private].
free Counter1:bitstring[private].
free Counter2:bitstring[private].
free Hello, HelloAck, SecureKeyBitstring:bitstring[private].
free KeyA:key [private].
free KeyB:key[private].
free SecureKey:seckey[private].
free Place_Holder:bitstring[private].
free EncryptedData:nonce[private].
free EncryptedData2:nonce[private].
free DecryptedData:nonce[private].
free DecryptedData2:nonce[private].
free DecryptedData4:bitstring[private].
free DecryptedData5:bitstring[private].
free DecryptedCMACData1:bitstring[private].
free DecryptedCMACData2:bitstring[private].
free DecryptedCMACData3:bitstring[private].
free DecryptedCMACData4:bitstring[private].

fun KeyBasedEnc(nonce, key): nonce.
reduc forall x: nonce, y: key; KeyBasedDecryption(KeyBasedEnc(x,y),y) = x.
fun SecureKeyEncryption(bitstring, seckey): nonce.
reduc forall x: bitstring, y: seckey; SecureKeyDecryption(
   SecureKeyEncryption(x,y),y) = x.

fun bitstringTOseckey(bitstring):seckey.

event TimerStart1_1(bitstring).
event TimerStart1_2(bitstring).
event TimerStart1_3(bitstring).
event TimerStart1_4(bitstring).
event TimerStart2_1(bitstring).
event TimerStart2_2(bitstring).
event TimerStart2_3(bitstring).
event TimerStart2_4(bitstring).
event DataDecryption1(nonce).
event DataDecryption2(nonce).
event DataDecryption3(nonce).
event DataDecryption4(bitstring).
event DataDecryption5(bitstring).
event CMACDataDecryption1(bitstring).
event CMACDataDecryption2(bitstring).
event CMACDataDecryption3(bitstring).
event CMACDataDecryption4(bitstring).
event CMACDataDecryption5(bitstring).
event DecryptedSecureKey(seckey).

let DeviceA1 =
  new RND_NumberA:bitstring;
  new Counter1:bitstring;
  new KeyA:key;
  let Counter1 = Counter1 in
    event TimerStart1_1(Counter1);
  out(c, CMAC(Data, RND_NumberA, Place_Holder, N, KeyA)).

let DeviceB1 =
  in(c, CMACData1:nonce);
  new KeyB:key;
  let DecryptedCMACData1 = DeCMAC(CMACData1, KeyB) in
    event CMACDataDecryption1(DecryptedCMACData1);
  new RND_NumberB:bitstring;
  new Counter2:bitstring;
  let Counter2 = Counter2 in
    event TimerStart2_1(Counter2);
  out(c, CMAC(Data, RND_NumberA, RND_NumberB, N, KeyB)).

let DeviceA2 =
  in(c, CMACData2:nonce);
  if Counter2 <> Counter2 then
    (
      0
      )
    else
    (
      let DecryptedCMACData2 = DeCMAC(CMACData2, KeyA) in
        event CMACDataDecryption2(DecryptedCMACData2);
      new Counter1:bitstring;
      let Counter1 = Counter1 in
        event TimerStart1_2(Counter1);
      out(c, KeyBasedEnc(CMAC(Data, RND_NumberA, RND_NumberB, N, KeyA),
          KeyA))
    ).

let DeviceB2 =
  in(c, EncryptedData:nonce);
  if Counter2 <> Counter2 then
  (
    0
  )
  else
  (
    new Counter2:bitstring;
    let Counter2 = Counter2 in
      event TimerStart2_2(Counter2);
    let DecryptedData = KeyBasedDecryption(EncryptedData, KeyB) in
      event DataDecryption1(DecryptedData);
    let DecryptedCMACData3 = DeCMAC(DecryptedData, KeyB) in
      event CMACDataDecryption3(DecryptedCMACData3);
    out(c, KeyBasedEnc(CMAC(Data, RND_NumberA, RND_NumberB, N, KeyB),KeyB)
       )
  ).


let DeviceA3 =
  in(c, EncryptedData2:nonce);
  if Counter1 <> Counter1 then
  (
    0
  )
  else
  (
    new Counter1:bitstring;
    let Counter1 = Counter1 in
      event TimerStart1_3(Counter1);
    let DecryptedData2 = KeyBasedDecryption(EncryptedData2, KeyA) in
      event DataDecryption2(DecryptedData2);
    let DecryptedCMACData4 = DeCMAC(DecryptedData2, KeyA) in
      event CMACDataDecryption4(DecryptedCMACData4);
    new SecureKeyBitstring:bitstring;
    new SecureKey:seckey;
    out(c, KeyBasedEnc(CMAC(SecureKeyBitstring, RND_NumberA, RND_NumberB,
        N, KeyA),KeyA))
  ).

  let DeviceB3 =
    in(c, EncryptedData3:nonce);
    if Counter2 <> Counter2 then
    (
      0
    )
    else
    (
      new Counter2:bitstring;
      let Counter2 = Counter2 in
        event TimerStart2_3(Counter2);
      let DecryptedData3 = KeyBasedDecryption(EncryptedData3, KeyB) in
        event DataDecryption3(DecryptedData3);
      let DecryptedCMACData5 = DeCMAC(DecryptedData3, KeyA) in
        event CMACDataDecryption5(DecryptedCMACData5);
      let SecureKey = bitstringTOseckey(DecryptedCMACData5) in
        event DecryptedSecureKey(SecureKey);
      out(c, SecureKeyEncryption(Hello, SecureKey))
      ).

    let DeviceA4 =
      in(c, EncryptedData4:nonce);
      if Counter1 <> Counter1 then
      (
        0
      )
      else
      (
        new Counter1:bitstring;
        let Counter1 = Counter1 in
          event TimerStart1_4(Counter1);
        let DecryptedData4 = SecureKeyDecryption(EncryptedData4, SecureKey
           ) in
          event DataDecryption4(DecryptedData4);
        out(c, SecureKeyEncryption(HelloAck, SecureKey))
      ).

      let DeviceB4 =
        in(c, EncryptedData5:nonce);
        if Counter2 <> Counter2 then
        (
          0
        )
        else
        (
          new Counter2:bitstring;
          let Counter2 = Counter2 in
            event TimerStart2_4(Counter2);
          let DecryptedData5 = SecureKeyDecryption(EncryptedData5,
              SecureKey) in
            event DataDecryption5(DecryptedData5)
        ).
```

```
process
(
    !DeviceA1|!DeviceB1|!DeviceA2|!DeviceB2|!DeviceA3|!DeviceA4|!DeviceB4
  )
```

Mutual Authentication ProVerif Result

```
Process:
(
    {1}!
    {2}new RND_NumberA_34: bitstring;
    {3}new Counter1_35: bitstring;
    {4}new KeyA_36: key;
    {5}let Counter1_37: bitstring = Counter1_35 in
    {6}event TimerStart1_1(Counter1_37);
    {7}out(c, CMAC(Data,RND_NumberA_34,Place_Holder,N,KeyA_36))
) | (
    {8}!
    {9}in(c, CMACData1_38: nonce);
    {10}new KeyB_39: key;
    {11}let DecryptedCMACData1_40: bitstring = DeCMAC(CMACData1_38,KeyB_39
       ) in
    {12}event CMACDataDecryption1(DecryptedCMACData1_40);
    {13}new RND_NumberB_41: bitstring;
    {14}new Counter2_42: bitstring;
    {15}let Counter2_43: bitstring = Counter2_42 in
    {16}event TimerStart2_1(Counter2_43);
    {17}out(c, CMAC(Data,RND_NumberA,RND_NumberB_41,N,KeyB_39))
) | (
    {18}!
    {19}in(c, CMACData2_44: nonce);
    {20}if (Counter2 <> Counter2) then
        0
    else
        {21}let DecryptedCMACData2_45: bitstring = DeCMAC(CMACData2_44,
           KeyA) in
        {22}event CMACDataDecryption2(DecryptedCMACData2_45);
        {23}new Counter1_46: bitstring;
        {24}let Counter1_47: bitstring = Counter1_46 in
        {25}event TimerStart1_2(Counter1_47);
        {26}out(c, KeyBasedEnc(CMAC(Data,RND_NumberA,RND_NumberB,N,KeyA),
           KeyA))
) | (
    {27}!
    {28}in(c, EncryptedData_48: nonce);
    {29}if (Counter2 <> Counter2) then
        0
    else
        {30}new Counter2_49: bitstring;
        {31}let Counter2_50: bitstring = Counter2_49 in
        {32}event TimerStart2_2(Counter2_50);
        {33}let DecryptedData_51: nonce = KeyBasedDecryption(
           EncryptedData_48,KeyB) in
        {34}event DataDecryption1(DecryptedData_51);
        {35}let DecryptedCMACData3_52: bitstring = DeCMAC(DecryptedData_51
           ,KeyB) in
        {36}event CMACDataDecryption3(DecryptedCMACData3_52);
        {37}out(c, KeyBasedEnc(CMAC(Data,RND_NumberA,RND_NumberB,N,KeyB),
           KeyB))
) | (
    {38}!
    {39}in(c, EncryptedData2_53: nonce);
    {40}if (Counter1 <> Counter1) then
        0
    else
        {41}new Counter1_54: bitstring;
        {42}let Counter1_55: bitstring = Counter1_54 in
        {43}event TimerStart1_3(Counter1_55);
        {44}let DecryptedData2_56: nonce = KeyBasedDecryption(
           EncryptedData2_53,KeyA) in
        {45}event DataDecryption2(DecryptedData2_56);
        {46}let DecryptedCMACData4_57: bitstring = DeCMAC(
           DecryptedData2_56,KeyA) in
        {47}event CMACDataDecryption4(DecryptedCMACData4_57);
        {48}new SecureKeyBitstring_58: bitstring;
        {49}new SecureKey_59: seckey;
        {50}out(c, KeyBasedEnc(CMAC(SecureKeyBitstring_58,RND_NumberA,
           RND_NumberB,N,KeyA),KeyA))
) | (
    {51}!
    {52}in(c, EncryptedData4: nonce);
    {53}if (Counter1 <> Counter1) then
        0
    else
        {54}new Counter1_60: bitstring;
        {55}let Counter1_61: bitstring = Counter1_60 in
        {56}event TimerStart1_4(Counter1_61);
        {57}let DecryptedData4_62: bitstring = SecureKeyDecryption(
           EncryptedData4,SecureKey) in
        {58}event DataDecryption4(DecryptedData4_62);
        {59}out(c, SecureKeyEncryption(HelloAck,SecureKey))
) | (
    {60}!
    {61}in(c, EncryptedData5: nonce);
    {62}if (Counter2 <> Counter2) then
        0
    else
        {63}new Counter2_63: bitstring;
        {64}let Counter2_64: bitstring = Counter2_63 in
        {65}event TimerStart2_4(Counter2_64);
        {66}let DecryptedData5_65: bitstring = SecureKeyDecryption(
           EncryptedData5,SecureKey) in
        {67}event DataDecryption5(DecryptedData5_65)
)

-- Query not attacker(SecureKey[])
nounif attacker(KeyBasedEnc(CMAC(DecryptedCMACData4_478,y_479,v_480,z_481,
   KeyA[]),KeyA[]))/-5000
Completing...
Starting query not attacker(SecureKey[])
RESULT not attacker(SecureKey[]) is true.
-- Query not attacker(SecureKeyBitstring[])
nounif attacker(KeyBasedEnc(CMAC(DecryptedCMACData4_1064,y_1065,v_1066,
   z_1067,KeyA[]),KeyA[]))/-5000
Completing...
Starting query not attacker(SecureKeyBitstring[])
RESULT not attacker(SecureKeyBitstring[]) is true.
-- Query not attacker(DecryptedData4[])
nounif attacker(KeyBasedEnc(CMAC(DecryptedCMACData4_1652,y_1653,v_1654,
   z_1655,KeyA[]),KeyA[]))/-5000
Completing...
Starting query not attacker(DecryptedData4[])
RESULT not attacker(DecryptedData4[]) is true.
-- Query not attacker(DecryptedData5[])
nounif attacker(KeyBasedEnc(CMAC(DecryptedCMACData4_2240,y_2241,v_2242,
   z_2243,KeyA[]),KeyA[]))/-5000
Completing...
Starting query not attacker(DecryptedData5[])
RESULT not attacker(DecryptedData5[]) is true.
-- Query not event(DataDecryption4(x_66))
nounif attacker(KeyBasedEnc(CMAC(DecryptedCMACData4_2828,y_2829,v_2830,
    z_2831,KeyA[]),KeyA[]))/-5000
Completing...
Starting query not event(DataDecryption4(x_66))
RESULT not event(DataDecryption4(x_66)) is true.
-- Query not event(DataDecryption5(x_67))
nounif attacker(KeyBasedEnc(CMAC(DecryptedCMACData4_3421,y_3422,v_3423,
    z_3424,KeyA[]),KeyA[]))/-5000
Completing...
Starting query not event(DataDecryption5(x_67))
RESULT not event(DataDecryption5(x_67)) is true.
-- Query event(DataDecryption4(x_68)) ==> event(DataDecryption5(x_68))
nounif attacker(KeyBasedEnc(CMAC(DecryptedCMACData4_4014,y_4015,v_4016,
    z_4017,KeyA[]),KeyA[]))/-5000
Completing...
Starting query event(DataDecryption4(x_68)) ==> event(DataDecryption5(x_68
    ))
RESULT event(DataDecryption4(x_68)) ==> event(DataDecryption5(x_68)) is
    true.
-- Query inj-event(DataDecryption4(x_69)) ==> inj-event(DataDecryption5(
    x_69))
nounif attacker(KeyBasedEnc(CMAC(DecryptedCMACData4_4607,y_4608,v_4609,
    z_4610,KeyA[]),KeyA[]))/-5000
Completing...
Starting query inj-event(DataDecryption4(x_69)) ==> inj-event(
    DataDecryption5(x_69))
RESULT inj-event(DataDecryption4(x_69)) ==> inj-event(DataDecryption5(x_69
    )) is true..
```

Unilateral Authentication ProVerif Verification

```
set traceDisplay = long.

query attacker(SecureKey).
query attacker (DeCMACDATA).
query attacker (DecryptedHello).
query x:seckey; event(DecryptedSecureKey(x)).
query x:bitstring; event(ConnectionVerification2(x)).
query x:seckey, y:bitstring; inj−event(DecryptedSecureKey(x)) ==> inj−
   event(ConnectionVerification2(y)).
query x:seckey, y:bitstring; event(DecryptedSecureKey(x)) ==> event(
   ConnectionVerification2(y)).

free c:channel.
type nonce.
type key.
type seckey.
type kkey.

free Data:bitstring [private].
free RND_NumberB:bitstring [private].
free Counter1:bitstring[private].
free Counter2:bitstring[private].
free KeyA:key [private].
free KeyB:key[private].
free KeyAA:kkey [private].
free KeyBB:kkey[private].
free DeCMACDATA:bitstring [private].
free DeCMACDATA2:bitstring [private].
free StartData:nonce [private].
free Hello:nonce[private].
free DecryptedHello, InTeachHello:bitstring[private].
free EncryptedHello:nonce[private].
free SecureKey:seckey[private].
free EncryptedData:nonce[private].
free DecryptedData:bitstring[private].

fun KeyBasedEnc(nonce, key): nonce.
reduc forall x: nonce, y: key; KeyBasedDecryption(KeyBasedEnc(x,y),y) = x.

fun SecureEncryption(bitstring, seckey): nonce.
```

```
reduc forall x: bitstring, y: seckey; SecureDecryption(SecureEncryption(x,
   y),y) = x.

fun CMAC(bitstring, bitstring, kkey): nonce.
reduc forall x: bitstring, y:bitstring, z:kkey; DeCMAC(CMAC(x,y,z),z) = x.

fun bitstringTOseckey(bitstring):seckey.

event TimerStart1(bitstring).
event TimerStart2(bitstring).
event DataDecryption1(nonce).
event RecievedData(bitstring).
event Connection(nonce).
event ConnectionVerification(nonce).
event ConnectionVerification2(bitstring).
event DecryptedSecureKey(seckey).

let DeviceA1 =
  new KeyA:key;
  out(c, Data).

let DeviceB1 =
  in(c, Data:bitstring);
  new KeyB:key;
  new RND_NumberB:bitstring;
  new Counter1:bitstring;
  let Counter1 = Counter1 in
    event TimerStart1(Counter1);
  out(c, (Data, RND_NumberB)).

let DeviceA2 =
  in(c, (Data:bitstring, RND_NumberB:bitstring));
  new Counter2:bitstring;
  let Counter2 = Counter2 in
    event TimerStart2(Counter2);
  if Counter2 = Counter2 then
  (
    new SecureKey:seckey;
    new SecureKeyBitstring:bitstring;
    out(c, KeyBasedEnc(CMAC(SecureKeyBitstring, RND_NumberB, KeyAA),KeyA))
    )
    else
    (
      0
      ).

let DeviceB2 =
  in(c, EncryptedData:nonce);
  if Counter1 = Counter1 then
  (
    let DecryptedData = KeyBasedDecryption(EncryptedData, KeyB) in
      event DataDecryption1(DecryptedData);
    let DeCMACDATA = DeCMAC(DecryptedData, KeyBB) in
      event RecievedData(DeCMACDATA);
    let SecureKey = bitstringTOseckey(DeCMACDATA) in
      event DecryptedSecureKey(SecureKey);
    out(c, SecureEncryption(InTeachHello, SecureKey))
  )
  else
  (
    0
  ).

  let DeviceA3 =
  in(c, EncryptedHello:nonce);
  if Counter2 = Counter2 then
  (
    let DecryptedHello = SecureDecryption(EncryptedHello, SecureKey) in
      event ConnectionVerification2(DecryptedHello)
    )
    else
    (
      0
      ).



process
(
    !DeviceA1
    |
    !DeviceB1
    |
    !DeviceA2
    |
    !DeviceB2
    |
    !DeviceA3

)
```

Unilateral Authentication ProVerif Result

```
Process:
(
    {1}!
    {2}new KeyA_32: key;
    {3}out(c, Data)
) | (
    {4}!
    {5}in(c, Data_33: bitstring);
    {6}new KeyB_34: key;
    {7}new RND_NumberB_35: bitstring;
    {8}new Counter1_36: bitstring;
    {9}let Counter1_37: bitstring = Counter1_36 in
    {10}event TimerStart1(Counter1_37);
    {11}out(c, (Data_33,RND_NumberB_35))
) | (
    {12}!
    {13}in(c, (Data_38: bitstring,RND_NumberB_39: bitstring));
    {14}new Counter2_40: bitstring;
    {15}let Counter2_41: bitstring = Counter2_40 in
    {16}event TimerStart2(Counter2_41);
    {17}if (Counter2_41 = Counter2_41) then
    {18}new SecureKey_42: seckey;
    {19}new SecureKeyBitstring: bitstring;
    {20}out(c, KeyBasedEnc(CMAC(SecureKeyBitstring,RND_NumberB_39,KeyAA),
       KeyA))
) | (
    {21}!
    {22}in(c, EncryptedData_43: nonce);
    {23}if (Counter1 = Counter1) then
    {24}let DecryptedData_44: nonce = KeyBasedDecryption(EncryptedData_43,
       KeyB) in
    {25}event DataDecryption1(DecryptedData_44);
    {26}let DeCMACDATA_45: bitstring = DeCMAC(DecryptedData_44,KeyBB) in
    {27}event RecievedData(DeCMACDATA_45);
    {28}let SecureKey_46: seckey = bitstringTOseckey(DeCMACDATA_45) in
    {29}event DecryptedSecureKey(SecureKey_46);
    {30}out(c, SecureEncryption(InTeachHello,SecureKey_46))
) | (
    {31}!
    {32}in(c, EncryptedHello_47: nonce);
    {33}if (Counter2 = Counter2) then
    {34}let DecryptedHello_48: bitstring = SecureDecryption(
        EncryptedHello_47,SecureKey) in
    {35}event ConnectionVerification2(DecryptedHello_48)
)

-- Query not attacker(SecureKey[])
Completing...
Starting query not attacker(SecureKey[])
RESULT not attacker(SecureKey[]) is true.
-- Query not attacker(DeCMACDATA[])
Completing...
Starting query not attacker(DeCMACDATA[])
RESULT not attacker(DeCMACDATA[]) is true.
-- Query not attacker(DecryptedHello[])
Completing...
Starting query not attacker(DecryptedHello[])
RESULT not attacker(DecryptedHello[]) is true.
-- Query not event(DecryptedSecureKey(x_49))
Completing...
Starting query not event(DecryptedSecureKey(x_49))
RESULT not event(DecryptedSecureKey(x_49)) is true.
-- Query not event(ConnectionVerification2(x_50))
Completing...
Starting query not event(ConnectionVerification2(x_50))
RESULT not event(ConnectionVerification2(x_50)) is true.
-- Query inj-event(DecryptedSecureKey(x_51)) ==> inj-event(
    ConnectionVerification2(y_52))
Completing...
Starting query inj-event(DecryptedSecureKey(x_51)) ==> inj-event(
    ConnectionVerification2(y_52))
RESULT inj-event(DecryptedSecureKey(x_51)) ==> inj-event(
    ConnectionVerification2(y_52)) is true.
-- Query event(DecryptedSecureKey(x_53)) ==> event(ConnectionVerification2
    (y_54))
Completing...
Starting query event(DecryptedSecureKey(x_53)) ==> event(
    ConnectionVerification2(y_54))
RESULT event(DecryptedSecureKey(x_53)) ==> event(ConnectionVerification2(
    y_54)) is true.
```

Appendix 3 – G.hn Authentication

Authentication is done according to the Diffie-Hellman algorithm and the Counter with Cipher Block Chaining-Message Authentication Code algorithm (CCM), which uses AES-128 standard for encryption. [82] This method is used to exchange cryptographic keys over public channel securely, without revealing one's private key to the public, and using mathematical operations to calculate another key to communicate with one another.

Diffie-Hellman algorithm works as follows:

Device A generates a random number R_A and sends it over to device B. Device B does exactly the same, generates a random number R_B and sends it to A. This is called a public key exchange. Both of these devices have their own private keys, which are used to to calculate session key K. Most common example is to do this with certificates, where devices derive public key from certificate.

CCM authentication is described in Appendix 19.

ProVerif Verification

```
set traceDisplay = long.

free c:channel.

type nonce.
type pubkey.
type privkey.
type key.
type certificate.
type seckey.

query attacker(ClientMKey).
query attacker(ServerMKey).
query attacker(DecryptedData).
query attacker(DecryptedData2).

query x:bitstring; event(Decryption1(x)).
query x:bitstring; event(Decryption2(x)).
query x:bitstring; event(Decryption1(x)) ==> event(Decryption2(x)).
query x:bitstring; inj-event(Decryption1(x)) ==> inj-event(Decryption2(x))
    .

free ServerCertificate:certificate[private].
free ClientCertificate:certificate[private].
free ClientPublicKey:key [private].
free ClientPrivateKey:key [private].
free ServerPublicKey:key[private].
free ServerPrivateKey:key [private].
free KeyExchanceVerification, ServerHello:bitstring[private].
free DecryptedVerification:bitstring[private].
free DecryptedVerificationACK:bitstring[private].
free Hello, Hello2, Hello2Ack:bitstring [private].
free KeyExchangeVerificationACK:bitstring[private].
fun PublicKeyGeneration(certificate):key.
fun PrivateKey(privkey):key.

free DecryptedData, DecryptedData2 :nonce [private].

type mkey.
free ServerMKey:mkey [private].
free ClientMKey:mkey [private].
fun MKeyGeneration(bitstring, key, key):mkey.
event ClientMKeyGeneration(mkey).
event ServerMKeyGeneration(mkey).

fun Encryption(bitstring, mkey):nonce.
reduc forall x:bitstring, y: mkey; Decryption(Encryption(x,y), y) = x.


event ClientPublicKeyGeneration(key).
event ServerPublicKeyGeneration(key).
event Decryption1(bitstring).
event Decryption2(bitstring).

let Client1 =
  out(c, Hello).

let Server1 =
  in(c, Hello:bitstring);
  out(c, (ServerHello, ServerCertificate)).

  let Client2 =
    in(c,(ServerHello:bitstring, ServerCertificate:certificate));
    let ServerPublicKey = PublicKeyGeneration(ServerCertificate) in
      event ServerPublicKeyGeneration(ServerPublicKey);
    let ClientMKey = MKeyGeneration(Hello, ServerPublicKey,
       ClientPrivateKey) in
      event ClientMKeyGeneration(ClientMKey);
    out(c, ClientCertificate).

  let Server2 =
    in(c, ClientCertificate:certificate);
    let ClientPublicKey = PublicKeyGeneration(ClientCertificate) in
      event ClientPublicKeyGeneration(ClientPublicKey);
    let ServerMKey = MKeyGeneration(ServerHello, ClientPublicKey,
       ServerPrivateKey) in
      event ServerMKeyGeneration(ServerMKey);
    out(c, Encryption(Hello2, ServerMKey)).

  let Client3 =
    in(c, EncryptedHello:nonce);
    let DecryptedData = Decryption(EncryptedHello, ClientMKey) in
```

```
      event Decryption1(DecryptedData);
    out(c, Encryption(Hello2Ack, ClientMKey)).

  let Server3 =
    in(c, EncryptedAck:nonce);
    let DecryptedData2 = Decryption(EncryptedAck, ServerMKey) in
      event Decryption2(DecryptedData2).

process
(
  !Client1|!Server1|!Client2|!Server2|!Client3|!Server3
  )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, Hello)
) | (
    {3}!
    {4}in(c, Hello_13: bitstring);
    {5}out(c, (ServerHello,ServerCertificate))
) | (
    {6}!
    {7}in(c, (ServerHello_14: bitstring,ServerCertificate_15: certificate)
       );
    {8}let ServerPublicKey_16: key = PublicKeyGeneration(
       ServerCertificate_15) in
    {9}event ServerPublicKeyGeneration(ServerPublicKey_16);
    {10}let ClientMKey_17: mkey = MKeyGeneration(Hello,ServerPublicKey_16,
       ClientPrivateKey) in
    {11}event ClientMKeyGeneration(ClientMKey_17);
    {12}out(c, ClientCertificate)
) | (
    {13}!
    {14}in(c, ClientCertificate_18: certificate);
    {15}let ClientPublicKey_19: key = PublicKeyGeneration(
       ClientCertificate_18) in
    {16}event ClientPublicKeyGeneration(ClientPublicKey_19);
    {17}let ServerMKey_20: mkey = MKeyGeneration(ServerHello,
       ClientPublicKey_19,ServerPrivateKey) in
    {18}event ServerMKeyGeneration(ServerMKey_20);
    {19}out(c, Encryption(Hello2,ServerMKey_20))
) | (
    {20}!
    {21}in(c, EncryptedHello: nonce);
    {22}let DecryptedData_21: bitstring = Decryption(EncryptedHello,
       ClientMKey) in
    {23}event Decryption1(DecryptedData_21);
    {24}out(c, Encryption(Hello2Ack,ClientMKey))
) | (
    {25}!
    {26}in(c, EncryptedAck: nonce);
    {27}let DecryptedData2_22: bitstring = Decryption(EncryptedAck,
        ServerMKey) in
    {28}event Decryption2(DecryptedData2_22)
)

-- Query not attacker(ClientMKey[])
Completing...
Starting query not attacker(ClientMKey[])
RESULT not attacker(ClientMKey[]) is true.
-- Query not attacker(ServerMKey[])
Completing...
Starting query not attacker(ServerMKey[])
RESULT not attacker(ServerMKey[]) is true.
-- Query not attacker(DecryptedData[])
Completing...
Starting query not attacker(DecryptedData[])
RESULT not attacker(DecryptedData[]) is true.
-- Query not attacker(DecryptedData2[])
Completing...
Starting query not attacker(DecryptedData2[])
RESULT not attacker(DecryptedData2[]) is true.
-- Query not event(Decryption1(x_23))
Completing...
Starting query not event(Decryption1(x_23))
RESULT not event(Decryption1(x_23)) is true.
-- Query not event(Decryption2(x_24))
Completing...
Starting query not event(Decryption2(x_24))
RESULT not event(Decryption2(x_24)) is true.
-- Query event(Decryption1(x_25)) ==> event(Decryption2(x_25))
Completing...
Starting query event(Decryption1(x_25)) ==> event(Decryption2(x_25))
RESULT event(Decryption1(x_25)) ==> event(Decryption2(x_25)) is true.
-- Query inj-event(Decryption1(x_26)) ==> inj-event(Decryption2(x_26))
Completing...
Starting query inj-event(Decryption1(x_26)) ==> inj-event(Decryption2(x_26
   ))
RESULT inj-event(Decryption1(x_26)) ==> inj-event(Decryption2(x_26)) is
   true.
```

Appendix 4 – KNX Authentication

KNX supports basic access protection to authenticate unicast communication, this allows to define up to 255 different access levels, where 0 is highest. Each access level can be secured by a different 4 byte password.

In this method, KNXnet/IP authentication is evaluated, because of these device are controlling the network. KNXnet/IP Secure is a security extension for KNXnet/IP that aims to be backward compatible. This means that no changes to the underlying KNX and KNXnet/IP protocol stack are required. The KNXnet/IP traffic is encapsulated in KNXnet/IP Secure wrapper which provides confidentiality, integrity, freshness and authenticity. KNXnet/IP encapsulated and encapsulates three logical layers in UDP/IP datagram and uses Secure Advanced Encryption Standard (AES) with 128 bit as a block cipher for all modes of operation. For authentication secret keys (Dk, Gk, pwd) are used. There are two types of communication in KNXnet/IP Secure, namely unicast communication and multicast communication. [144]

Multicast communication secures the traffic between members of one group. The mode of operation uses a modified version of CCM. This communication type uses a pre-shared secret called Group key (Gk). This key is unique for every multicast group. The same group key can be found on every device that is in the same group The group key has a size of 128 bit. [144]

Unicast traffic is used for configuration purposes, this means that securing the communication between a management device and an interconnection device is needed. To achieve secrecy, KNXnet/IP Secure uses Elliptic Curve Diffie Hellman (ECDH) key exchange algorithm over NIST curve. To authenticate the communication, two pre-shared secret keys are used Device authentication code (Dk) and Passwords (Pwd). In the ECDH key exchange, Dk serves as public key and Pwd as private key. [144]

Diffie-Hellman authentication has been described in Appendix 3

ProVerif Verification

```
type nonce.
type certificate.
type key.
type ssecret.
query attacker(DeviceKeyA).
query attacker(DeviceKeyB).
query attacker(Password).
query attacker(PrivateDeviceKeyA).
query attacker(PrivateDeviceKeyB).
query attacker(sharedsecret).
query attacker (TESTDecrypt1).
query attacker (TESTDecrypt2).
query x:bitstring; event(TESTDecryption1(x)).
query x:bitstring; event(TESTDecryption2(x)).
query x:bitstring; event(TESTDecryption1(x)) ==> event(TESTDecryption2(x))
    .
query x:bitstring; inj-event(TESTDecryption1(x)) ==> inj-event(
    TESTDecryption2(x)).

free c:channel.

free sharedsecret:ssecret [private].
free DeviceCertA:certificate [private].
free DeviceCertB:certificate[private].
free DeviceKeyA:key[private].
free PublicDeviceKeyA:key[private].
free PrivateDeviceKeyA:key [private].
free DeviceKeyB:key[private].
free PublicDeviceKeyB:key[private].
free PrivateDeviceKeyB:key[private].
free PasswordFlag:bitstring[private].
free Password:key [private].
free TESTDecrypt1:bitstring [private].
free TESTDecrypt2:bitstring [private].

fun CTR(nonce,ssecret):nonce.
reduc forall x: nonce, y: ssecret; DeCTR(CTR(x,y),y) = x.
fun CBC(nonce, key): nonce.
reduc forall x: nonce, y:key ; DeCBC(CBC(x,y), y) = x.
fun TCNonce(bitstring):nonce.
fun TCKey(nonce):key.
fun TCssecret(ssecret):key.
fun TCCertificate(certificate):nonce.
fun CCM(bitstring, nonce, ssecret):nonce.
reduc forall z: bitstring, x: nonce, y:ssecret; DeCCM(CCM(z,x,y), y) = z.
fun KeyGeneration(certificate):key.

event DeviceAPublicKeyGeneration(key).
event DeviceBPublicKeyGeneration(key).
event TESTDecryption1(bitstring).
event TESTDecryption2(bitstring).

let DeviceA1 =
  new Hello:bitstring;
  out(c,Hello).

let DeviceB1 =
  in(c, Hello:bitstring);
  new HelloACK:bitstring;
  out(c, HelloACK).

let DeviceA2 =
  in(c, HelloACK:bitstring);
  new PublicDeviceKeyA:key;
  out(c, DeviceCertA).

let DeviceB2 =
  in(c, DeviceCertA:certificate);
  let PublicDeviceKeyA = KeyGeneration(DeviceCertA) in
    event DeviceAPublicKeyGeneration(PublicDeviceKeyA);
  new DeviceCertBTemplate:bitstring;
  out(c, CTR(CBC(TCCertificate(DeviceCertB), DeviceKeyA), sharedsecret)).

let DeviceA3 =
  in(c, EncryptedResponse:nonce);
  let PublicDeviceKeyB = TCKey(DeCTR(DeCBC(EncryptedResponse,
     PrivateDeviceKeyA), sharedsecret)) in
    event DeviceBPublicKeyGeneration(PublicDeviceKeyB);
  new PublicDeviceKeyBTemplate:nonce;
  out(c, CCM(PasswordFlag, CBC(PublicDeviceKeyBTemplate,Password),
     sharedsecret)).
```

```
let DeviceB3 =
  in(c, EncryptedComplete:nonce);
  let TESTDecrypt1 = DeCCM(DeCBC(EncryptedComplete, TCssecret(sharedsecret
     )), sharedsecret) in
    event TESTDecryption1(TESTDecrypt1);
  new Success:bitstring;
  out(c, CCM(PasswordFlag, CBC(TCNonce(Success),Password), sharedsecret)).

let DeviceA4 =
  in(c, TestEncrypt:nonce);
  let TESTDecrypt2 = DeCCM(DeCBC(TestEncrypt,TCssecret(sharedsecret)),
     sharedsecret) in
    event TESTDecryption2(TESTDecrypt2).



process
  (
    !DeviceA1|!DeviceB1|!DeviceA2|!DeviceB2|!DeviceA3|!DeviceB3|!DeviceA4
    )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}new Hello: bitstring;
    {3}out(c, Hello)
) | (
    {4}!
    {5}in(c, Hello_29: bitstring);
    {6}new HelloACK: bitstring;
    {7}out(c, HelloACK)
) | (
    {8}!
    {9}in(c, HelloACK_30: bitstring);
    {10}new PublicDeviceKeyA_31: key;
    {11}out(c, DeviceCertA)
) | (
    {12}!
    {13}in(c, DeviceCertA_32: certificate);
    {14}let PublicDeviceKeyA_33: key = KeyGeneration(DeviceCertA_32) in
    {15}event DeviceAPublicKeyGeneration(PublicDeviceKeyA_33);
    {16}new DeviceCertBTemplate: bitstring;
    {17}out(c, CTR(CBC(TCCertificate(DeviceCertB),DeviceKeyA),sharedsecret
       ))
) | (
    {18}!
    {19}in(c, EncryptedResponse: nonce);
    {20}let PublicDeviceKeyB_34: key = TCKey(DeCTR(DeCBC(EncryptedResponse
       ,PrivateDeviceKeyA),sharedsecret)) in
    {21}event DeviceBPublicKeyGeneration(PublicDeviceKeyB_34);
    {22}new PublicDeviceKeyBTemplate: nonce;
    {23}out(c, CCM(PasswordFlag,CBC(PublicDeviceKeyBTemplate,Password),
       sharedsecret))
) | (
    {24}!
    {25}in(c, EncryptedComplete: nonce);
    {26}let TESTDecrypt1_35: bitstring = DeCCM(DeCBC(EncryptedComplete,
       TCssecret(sharedsecret)),sharedsecret) in
    {27}event TESTDecryption1(TESTDecrypt1_35);
    {28}new Success: bitstring;
    {29}out(c, CCM(PasswordFlag,CBC(TCNonce(Success),Password),
        sharedsecret))
) | (
    {30}!
    {31}in(c, TestEncrypt: nonce);
    {32}let TESTDecrypt2_36: bitstring = DeCCM(DeCBC(TestEncrypt,TCssecret
        (sharedsecret)),sharedsecret) in
    {33}event TESTDecryption2(TESTDecrypt2_36)
)

-- Query not attacker(DeviceKeyA[])
Completing...
Starting query not attacker(DeviceKeyA[])
RESULT not attacker(DeviceKeyA[]) is true.
-- Query not attacker(DeviceKeyB[])
Completing...
Starting query not attacker(DeviceKeyB[])
RESULT not attacker(DeviceKeyB[]) is true.
-- Query not attacker(Password[])
Completing...
Starting query not attacker(Password[])
RESULT not attacker(Password[]) is true.
-- Query not attacker(PrivateDeviceKeyA[])
Completing...
Starting query not attacker(PrivateDeviceKeyA[])
RESULT not attacker(PrivateDeviceKeyA[]) is true.
-- Query not attacker(PrivateDeviceKeyB[])
Completing...
Starting query not attacker(PrivateDeviceKeyB[])
RESULT not attacker(PrivateDeviceKeyB[]) is true.
-- Query not attacker(sharedsecret[])
Completing...
Starting query not attacker(sharedsecret[])
RESULT not attacker(sharedsecret[]) is true.
-- Query not attacker(TESTDecrypt1[])
Completing...
Starting query not attacker(TESTDecrypt1[])
RESULT not attacker(TESTDecrypt1[]) is true.
-- Query not attacker(TESTDecrypt2[])
Completing...
Starting query not attacker(TESTDecrypt2[])
RESULT not attacker(TESTDecrypt2[]) is true.
-- Query not event(TESTDecryption1(x_37))
```

```
Completing...
Starting query not event(TESTDecryption1(x_37))
RESULT not event(TESTDecryption1(x_37)) is true.
-- Query not event(TESTDecryption2(x_38))
Completing...
Starting query not event(TESTDecryption2(x_38))
RESULT not event(TESTDecryption2(x_38)) is true.
-- Query event(TESTDecryption1(x_39)) ==> event(TESTDecryption2(x_39))
Completing...
Starting query event(TESTDecryption1(x_39)) ==> event(TESTDecryption2(x_39
    ))
RESULT event(TESTDecryption1(x_39)) ==> event(TESTDecryption2(x_39)) is
    true.
-- Query inj-event(TESTDecryption1(x_40)) ==> inj-event(TESTDecryption2(
    x_40))
Completing...
Starting query inj-event(TESTDecryption1(x_40)) ==> inj-event(
    TESTDecryption2(x_40))
RESULT inj-event(TESTDecryption1(x_40)) ==> inj-event(TESTDecryption2(x_40
    )) is true.
```

Appendix 5 – WiMAX Authentication

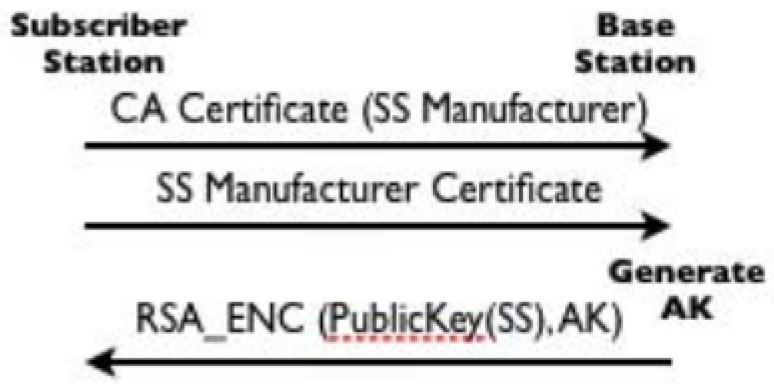
PKMv1

PKMv1 uses asymmetric ciphers (primarily RSA) to validate identity of stations. A subscriber station starts the authentication process with presentation of the station's certificate issued by its manufacturer. Optionally it may can send the hardware manufacturer certificate. Following certificate validation, the base station will generate an Authorization Key (AK) which is sent back to the subscriber station, encrypting it with a public key obtained from the subscriber station's certificate. The subscriber station decrypts it using its private key. In this way both parties obtain cryptographic material necessary for link protection. The PKMv1 protocol skips user data authentication ensuring only its privacy. [5]
Exchange of PKMv1 authentication messages has been shown in Figure 6.
The transmission Encryption Key (TEK), which is generated at a later stage, similarly to AK is determined by the base station. Each new association is assigned with an individual encryption key. The key, which is used to ensure secure transmission of the TEK key is the Key Encryption Key (KEK). KEK is generated based on the AK key as well as Hash Message Authentication Code Key for Downlink and Hash Message Authentication Code Key for Uplink keys. The purpose of the KEK is to encrypt messages that are used to transfer the TEK keys. On the other hand, HMAC family of keys are used to generate authentication checksums in control messages. These checksums are calculated using the SHA-1 function. Key transmission messages are encrypted with 3DES algorithm in a mode, in which each 64-bit data block is encrypted independently. [5]
It is worth mentioning here that each security association consists of two TEKs (the current and a spare one). Traffic decryption can be performed using both of these keys, whereas only the current key permits data encryption. For purposes of user protection, the PKMv1 protocol of the IEEE 802.16d standard utilizes advantages of the AES block algorithm operating in a special mode called Counter Mode with Cipher Block Chaining Message Authentication Code (CCM). [5]
CCM authentication is described in Appendix 19

PKMv2

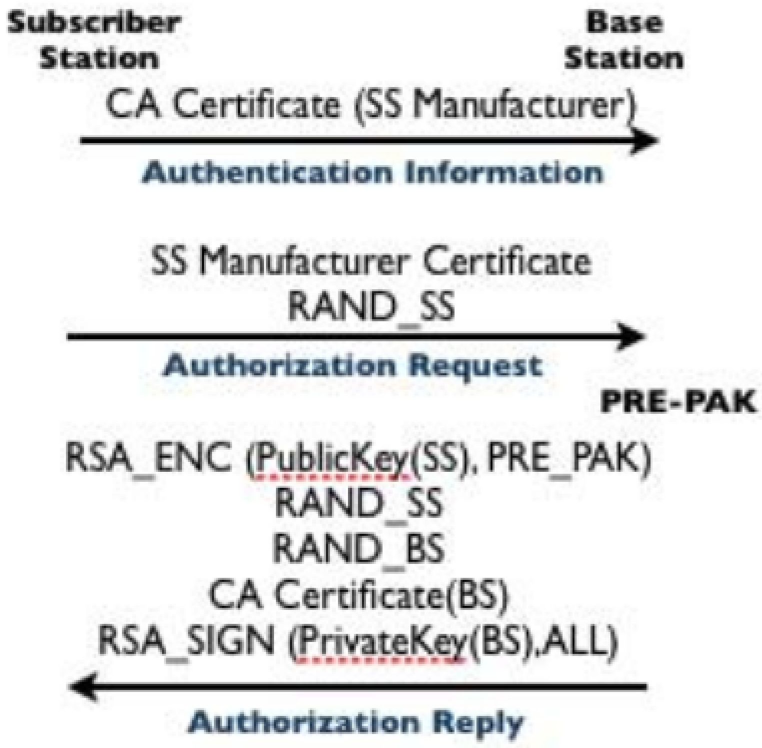
[75] Version 2 of the PKM protocol was developed because of the numerous shortcomings in the protection system used in the previous version of the protocol. The PKMv2

Figure 6: WiMax PKMv1 message flow for authentication [5]


Subscriber Station
Base Station
CA Certificate (SS Manufacturer)
SS Manufacturer Certificate
Generate AK
RSA_ENC (PublicKey(SS).AK)

standard bases on the techniques adopted in the IEEE 802.11i standard. Contrary to the first version of the protocol, the authentication stage in the second version has been implemented based on the Extensible Authentication Protocol (EAP). The network infrastructure has been also extended with an Authentication, Authorization, Accounting (AAA) server supporting the EAP on the network side. [5]
Exchange of PKMv2 authentication messages has been shown in Figure 7.
The result of these operations is a unique cryptographic material, generated independently by mutually authenticating parties. The DES and AES algorithms have been used to ensure link privacy. The PKMv2 protocol assumes that both subscriber and base stations should have a certificate.
Terminal authentication starts with optional presentation of a manufacturer's certificate to the base station. Then the base station sends an authentication request containing a certificate issued by its manufacturer and a generated random number (RAND_SS). Having verified the certificate, the base station responds with a message containing its own X.509 certificate encrypted with the subscriber station's public key (Pre-Primary Authorization Key /Pre-PAK/), the RAND_SS number received in the previous message and with its generated RAND_BS number. Next, the subscriber station verifies the validity of the certificate received, check correctness of the previously generated random number and deciphers the Pre-PAK key. [5]

Figure 7: WiMax PKMv2 message flow for authentication [5]


Subscriber Station
Base Station
CA Certificate (SS Manufacturer)
Authentication Information
SS Manufacturer Certificate
RAND_SS
Authorization Request
PRE-PAK
RSA_ENC (PublicKey(SS), PRE_PAK)
RAND_SS
RAND_BS
CA Certificate(BS)
RSA_SIGN (PrivateKey(BS),ALL)
Authorization Reply

WiMAX PKMv1 ProVerif Verification

```
set traceDisplay = long.

query attacker(privatekey).
query attacker(AK).
query attacker(AK2).
query attacker(DecryptedText1).
query attacker(DecryptedText2).
query x: authkey; event(AuthorizationKeyGen(x)).
query x: bitstring; event(TextDecryption1(x)).
query x: bitstring; event(TextDecryption2(x)).
query x:bitstring; event(TextDecryption1(x)) ==> event(TextDecryption2(x))
    .
query x:bitstring; inj-event(TextDecryption1(x)) ==> inj-event(
   TextDecryption2(x)).

free c: channel.

type certificate.
type authkey.
type key.
type nonce.
free Hello:bitstring[private].
free HelloAck:bitstring[private].
free StationCertificate: certificate [private].
free ManufacturerCertificate: certificate [private].
free AK: authkey [private].
free info:bitstring[private].
free pubkey: key [private].
free privatekey: key [private].
free DecryptedText1: bitstring [private].
free DecryptedText2: bitstring [private].

free ENCauthkey: nonce [private].

free AK2: authkey [private].

fun CertificateGen(bitstring, key): certificate.
reduc forall x: bitstring, y: key; dec_Certificate(CertificateGen(x,y), y)
    = x.
```

```
fun authkey_enc(authkey, key): nonce.
reduc forall x: authkey, y: key; dec_authkey(authkey_enc(x,y), y) = x.

fun Encryption(bitstring, authkey): nonce.
reduc forall x: bitstring, y: authkey; Decryption(Encryption(x,y),y)=x.

event CertificateValidation(certificate).
event AuthorizationKeyGen(authkey).
event Authdec(authkey).
event TextDecryption1(bitstring).
event TextDecryption2(bitstring).

let SubscriberStation1 =
        out(c, (StationCertificate, info)).

let BaseStation1 =
        in(c, (StationCertificate:certificate, info:bitstring));
        let StationCertificate = ManufacturerCertificate in
                event CertificateValidation(StationCertificate);
        if StationCertificate = ManufacturerCertificate then
        (
                new AKey:authkey;
                let AK = AKey in
                        event AuthorizationKeyGen(AKey);
                out(c, authkey_enc(AKey, pubkey))
        )
        else
        (
                0
        ).

let SubscriberStation2 =
        in(c, ENCauthkey:nonce);
        let AK2 = dec_authkey(ENCauthkey, privatekey) in
                event Authdec(AK2);
        out(c, Encryption(Hello, AK)).

let BaseStation2 =
        in(c, EncryptedText1:nonce);
        let DecryptedText1 = Decryption(EncryptedText1, AK) in
                event TextDecryption1(DecryptedText1);
        out(c, Encryption(HelloAck, AK)).
```

```
let SubscriberStation3 =
        in(c, EncryptedText2:nonce);
        let DecryptedText2 = Decryption(EncryptedText2, AK2) in
                event TextDecryption2(DecryptedText2).



process
(
        !SubscriberStation1
        |
        !BaseStation1
        |
        !SubscriberStation2
        |
        !BaseStation2
        |
        !SubscriberStation3
)
```

WiMAX PKMv1 ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, (StationCertificate,info))
) | (
    {3}!
    {4}in(c, (StationCertificate_32: certificate,info_33: bitstring));
    {5}let StationCertificate_34: certificate = ManufacturerCertificate in
    {6}event CertificateValidation(StationCertificate_34);
    {7}if (StationCertificate_34 = ManufacturerCertificate) then
    {8}new AKey: authkey;
    {9}let AK_35: authkey = AKey in
    {10}event AuthorizationKeyGen(AKey);
    {11}out(c, authkey_enc(AKey,pubkey))
) | (
    {12}!
    {13}in(c, ENCauthkey_36: nonce);
    {14}let AK2_37: authkey = dec_authkey(ENCauthkey_36,privatekey) in
    {15}event Authdec(AK2_37);
    {16}out(c, Encryption(Hello,AK))
) | (
    {17}!
    {18}in(c, EncryptedText1: nonce);
    {19}let DecryptedText1_38: bitstring = Decryption(EncryptedText1,AK)
       in
    {20}event TextDecryption1(DecryptedText1_38);
    {21}out(c, Encryption(HelloAck,AK))
) | (
    {22}!
    {23}in(c, EncryptedText2: nonce);
    {24}let DecryptedText2_39: bitstring = Decryption(EncryptedText2,AK2)
       in
    {25}event TextDecryption2(DecryptedText2_39)
)

-- Query not attacker(privatekey[])
Completing...
Starting query not attacker(privatekey[])
RESULT not attacker(privatekey[]) is true.
-- Query not attacker(AK[])
```

```
Completing...
Starting query not attacker(AK[])
RESULT not attacker(AK[]) is true.
-- Query not attacker(AK2[])
Completing...
Starting query not attacker(AK2[])
RESULT not attacker(AK2[]) is true.
-- Query not attacker(DecryptedText1[])
Completing...
Starting query not attacker(DecryptedText1[])
RESULT not attacker(DecryptedText1[]) is true.
-- Query not attacker(DecryptedText2[])
Completing...
Starting query not attacker(DecryptedText2[])
RESULT not attacker(DecryptedText2[]) is true.
-- Query not event(AuthorizationKeyGen(x_40))
Completing...
Starting query not event(AuthorizationKeyGen(x_40))
goal reachable: attacker(StationCertificate_1352) && attacker(info_1353)
    -> end(AuthorizationKeyGen(AKey[info_33 = info_1353,
    StationCertificate_32 = StationCertificate_1352,!1 = @sid_1354]))
Abbreviations:
AKey_1362 = AKey[info_33 = info_1359,StationCertificate_32 =
    StationCertificate_1358,!1 = @sid_1360]

1. We assume as hypothesis that
attacker(info_1359).

2. We assume as hypothesis that
attacker(StationCertificate_1358).

3. By 2, the attacker may know StationCertificate_1358.
By 1, the attacker may know info_1359.
Using the function 2-tuple the attacker may obtain (
    StationCertificate_1358,info_1359).
attacker((StationCertificate_1358,info_1359)).

4. The message (StationCertificate_1358,info_1359) that the attacker may
    have by 3 may be received at input {4}.
So event AuthorizationKeyGen(AKey_1362) may be executed at {10}.
end(AuthorizationKeyGen(AKey_1362)).
```

```
Initial state

Additional knowledge of the attacker:
c
a_1363
a
```

```
New processes:
    (
        !
        out(c, (StationCertificate,info))
    ) | (
        !
        in(c, (StationCertificate_32: certificate,info_33: bitstring));
        let StationCertificate_34: certificate = ManufacturerCertificate
           in
        event CertificateValidation(StationCertificate_34);
        if (StationCertificate_34 = ManufacturerCertificate) then
        new AKey: authkey;
        let AK_35: authkey = AKey in
        event AuthorizationKeyGen(AKey);
        out(c, authkey_enc(AKey,pubkey))
    ) | (
        !
        in(c, ENCauthkey_36: nonce);
        let AK2_37: authkey = dec_authkey(ENCauthkey_36,privatekey) in
        event Authdec(AK2_37);
        out(c, Encryption(Hello,AK))
    ) | (
        !
        in(c, EncryptedText1: nonce);
        let DecryptedText1_38: bitstring = Decryption(EncryptedText1,AK)
           in
        event TextDecryption1(DecryptedText1_38);
        out(c, Encryption(HelloAck,AK))
    ) | (
        !
        in(c, EncryptedText2: nonce);
        let DecryptedText2_39: bitstring = Decryption(EncryptedText2,AK2)
           in
        event TextDecryption2(DecryptedText2_39)
    )
```

```
1st process: Reduction |

2nd process: Reduction |

3rd process: Reduction |

4th process: Reduction |

5th process: Reduction ! 0 copy(ies)

4th process: Reduction ! 0 copy(ies)

3rd process: Reduction ! 0 copy(ies)

2nd process: Reduction ! 1 copy(ies)

2nd process: Beginning of process BaseStation1

1st process: Reduction ! 0 copy(ies)

New processes:
    in(c, (StationCertificate_1369: certificate,info_1370: bitstring));
    let StationCertificate_1371: certificate = ManufacturerCertificate in
    event CertificateValidation(StationCertificate_1371);
    if (StationCertificate_1371 = ManufacturerCertificate) then
    new AKey: authkey;
    let AK_1372: authkey = AKey in
    event AuthorizationKeyGen(AKey);
    out(c, authkey_enc(AKey,pubkey))
```

```
1st process: in(c, (StationCertificate_1369: certificate,info_1370:
   bitstring)) done with message (a_1363,a)

1st process: let StationCertificate_1379: certificate =
   ManufacturerCertificate succeeds

1st process: event CertificateValidation(ManufacturerCertificate) executed
```

```
1st process: if (ManufacturerCertificate = ManufacturerCertificate)
   succeeds

1st process: new AKey: authkey creating AKey_1365

1st process: let AK_1381: authkey = AKey_1365 succeeds

1st process: event AuthorizationKeyGen(AKey_1365) executed; it is a goal

New processes:
    out(c, authkey_enc(AKey_1365,pubkey))

```

```
The event AuthorizationKeyGen(AKey_1365) is executed.
A trace has been found.
RESULT not event(AuthorizationKeyGen(x_40)) is false.
-- Query not event(TextDecryption1(x_41))
Completing...
Starting query not event(TextDecryption1(x_41))
RESULT not event(TextDecryption1(x_41)) is true.
-- Query not event(TextDecryption2(x_42))
Completing...
Starting query not event(TextDecryption2(x_42))
RESULT not event(TextDecryption2(x_42)) is true.
-- Query event(TextDecryption1(x_43)) ==> event(TextDecryption2(x_43))
Completing...
Starting query event(TextDecryption1(x_43)) ==> event(TextDecryption2(x_43
   ))
RESULT event(TextDecryption1(x_43)) ==> event(TextDecryption2(x_43)) is
   true.
-- Query inj-event(TextDecryption1(x_44)) ==> inj-event(TextDecryption2(
   x_44))
Completing...
Starting query inj-event(TextDecryption1(x_44)) ==> inj-event(
   TextDecryption2(x_44))
RESULT inj-event(TextDecryption1(x_44)) ==> inj-event(TextDecryption2(x_44
   )) is true.
```

WiMAX PKMv2 ProVerif Verification

```
set traceDisplay = long.

query attacker (AuthorizationKey).
query attacker (SSPrivateKey).
query attacker (BSPrivateKey).
query attacker(DecryptedText1).
query attacker(DecryptedText2).
query x: bitstring; event(Decryption1(x)).
query x: bitstring; event(Decryption2(x)).
query x:bitstring; event(Decryption1(x)) ==> event(Decryption2(x)).
query x:bitstring; inj-event(Decryption1(x)) ==> inj-event(Decryption2(x))
    .


free c: channel.

type certificate.
type nonce.
type key.

free SSCertificate:certificate [private].
free N:nonce [private].
free SSPublicKey:key [private].
free SSPrivateKey:key [private].
free BSPrivateKey:key [private].
free BSCertificate:certificate [private].
free AuthorizationKey:certificate [private].
free ACK:bitstring [private].
free Hello:bitstring[private].
free HelloAck:bitstring[private].

fun CertSign(certificate, nonce) : nonce.
reduc forall x:certificate, y:nonce; DeSignCert(CertSign(x,y), y) = x.

fun CertPub(certificate, key) : nonce.
reduc forall x:certificate, y:key; DeCertPub(CertPub(x,y), y) = x.

fun Encrypt(bitstring, certificate) : nonce.
reduc forall x:bitstring, y: certificate; Decrypt(Encrypt(x,y),y) = x.
```

```
event AuthorizationKeyGeneration(certificate).
event CertificateGeneration(certificate).
event Decryption1(bitstring).
event Decryption2(bitstring).

let SS1 =
        new SSCertificate:certificate;
        let SSCertificate = SSCertificate in
                event CertificateGeneration(SSCertificate);
        out(c, SSCertificate).

let SS2 =
        out(c, CertSign(SSCertificate , N)).

let BS1 =
        in(c, SignedCertificate:nonce);
        if DeSignCert(SignedCertificate,N) = SSCertificate then
        (
                out(c,CertPub(BSCertificate,SSPublicKey))
        )
        else
        (
                0
        ).

let SS3 =
        in(c, SignedBSCertificate:nonce);
        let AuthorizationKey = DeCertPub(SignedBSCertificate , SSPrivateKey
           ) in
                event AuthorizationKeyGeneration(AuthorizationKey);
        out(c, Encrypt(Hello , AuthorizationKey)).

let BS2 =
        in(c, EncryptedText1 : nonce);
        let DecryptedText1 = Decrypt(EncryptedText1 , AuthorizationKey) in
                event Decryption1(DecryptedText1);
        out(c, Encrypt(HelloAck , AuthorizationKey)).

let SS4 =
        in(c, EncryptedText2 : nonce);
        let DecryptedText2  = Decrypt(EncryptedText2 , AuthorizationKey) in
                event Decryption2(DecryptedText2).
```

```
process
        (
                !SS1
                |
                !SS2
                |
                !BS1
                |
                !SS3
                |
                !BS2
                |
                !SS4
        )
```

WiMAX PKMv2 ProVerif Result

```
Process:
(
    {1}!
    {2}new SSCertificate_32: certificate;
    {3}let SSCertificate_33: certificate = SSCertificate_32 in
    {4}event CertificateGeneration(SSCertificate_33);
    {5}out(c, SSCertificate_33)
) | (
    {6}!
    {7}out(c, CertSign(SSCertificate,N))
) | (
    {8}!
    {9}in(c, SignedCertificate: nonce);
    {10}if (DeSignCert(SignedCertificate,N) = SSCertificate) then
    {11}out(c, CertPub(BSCertificate,SSPublicKey))
) | (
    {12}!
    {13}in(c, SignedBSCertificate: nonce);
    {14}let AuthorizationKey_34: certificate = DeCertPub(
       SignedBSCertificate,SSPrivateKey) in
    {15}event AuthorizationKeyGeneration(AuthorizationKey_34);
    {16}out(c, Encrypt(Hello,AuthorizationKey_34))
) | (
    {17}!
    {18}in(c, EncryptedText1: nonce);
    {19}let DecryptedText1_35: bitstring = Decrypt(EncryptedText1,
       AuthorizationKey) in
    {20}event Decryption1(DecryptedText1_35);
    {21}out(c, Encrypt(HelloAck,AuthorizationKey))
) | (
    {22}!
    {23}in(c, EncryptedText2: nonce);
    {24}let DecryptedText2_36: bitstring = Decrypt(EncryptedText2,
       AuthorizationKey) in
    {25}event Decryption2(DecryptedText2_36)
)

-- Query not attacker(AuthorizationKey[])
Completing...
Starting query not attacker(AuthorizationKey[])
```

```
RESULT not attacker(AuthorizationKey[]) is true.
-- Query not attacker(SSPrivateKey[])
Completing...
Starting query not attacker(SSPrivateKey[])
RESULT not attacker(SSPrivateKey[]) is true.
-- Query not attacker(BSPrivateKey[])
Completing...
Starting query not attacker(BSPrivateKey[])
RESULT not attacker(BSPrivateKey[]) is true.
-- Query not attacker(DecryptedText1[])
Completing...
Starting query not attacker(DecryptedText1[])
RESULT not attacker(DecryptedText1[]) is true.
-- Query not attacker(DecryptedText2[])
Completing...
Starting query not attacker(DecryptedText2[])
RESULT not attacker(DecryptedText2[]) is true.
-- Query not event(Decryption1(x_37))
Completing...
Starting query not event(Decryption1(x_37))
RESULT not event(Decryption1(x_37)) is true.
-- Query not event(Decryption2(x_38))
Completing...
Starting query not event(Decryption2(x_38))
RESULT not event(Decryption2(x_38)) is true.
-- Query event(Decryption1(x_39)) ==> event(Decryption2(x_39))
Completing...
Starting query event(Decryption1(x_39)) ==> event(Decryption2(x_39))
RESULT event(Decryption1(x_39)) ==> event(Decryption2(x_39)) is true.
-- Query inj-event(Decryption1(x_40)) ==> inj-event(Decryption2(x_40))
Completing...
Starting query inj-event(Decryption1(x_40)) ==> inj-event(Decryption2(x_40
   ))
RESULT inj-event(Decryption1(x_40)) ==> inj-event(Decryption2(x_40)) is
   true.
```

Appendix 6 – Z-Wave Authentication

Z-Wave uses Out-of-Band authentication to be able to join Z-Wave PAN network. OoB authentication is Z-Wave's solution to mitigate the inclusion of rogue nodes. [63]
Like Wi-Fi, Z-Wave's S2 security also operates with a network key. A strong temporary key is used to assign keys for security classes. This allows segmentation of safety critical devices like S2 Access Control and S2 Authenticated class. S2 Unauthenticated class is used for constrained devices without authentication. Access to Z-Wave networks is controlled by Z-Wave gateway, which forwards commands only from already trusted LAN clients or trusted ISP. DTLS is used to secure communication between LAN hosts and Z-Wave nodes. LAN hosts and Z-Wave nodes communicate via a Z/IP Gateway which terminates the DTLS encryption and strips Z/IP and IP headers before forwarding Z-Wave commands securely in the Z-Wave network. [136] DTLS Authentication is described in Appendix 10. A given S2 security class not only identifies the network key to use but also dictates the rules applying to authentication of a new node during inclusion. [136]
Z-Wave Authentication sequence has been show in Figure 8.
The "S2 Access Control" class is the highest trusted class, intended only for access control devices like doors and locks. [136]
The "S2 Authenticated" class is used for regular household devices such as sensors and lights. The "S2 Unauthenticated" class is the least trusted class and is intended for the most constrained controllers that are not capable of authenticating a device joining the network. [136]
S0 and all three S2 classes use AES-128 encryption. A node, which has requested and been granted access to several security classes during inclusion but it only accepts incoming commands from the trusted classes. This means that if light bulb is a member of both the S2 Authenticated and S2 Unauthenticated classes, it only accepts commands encrypted with the S2 Authenticated class key, because its security requirements are the highest.
The S2 authentication process allows a controller to verify that a joining node is device that it claims to be. Depending on the UI, a controller may allow the installer to enter a Device-Specific Key (DSK). The DSK is the first 16 bytes of the 32-byte long Elliptic Curve Diffie-Hellman key exchange (ECDH) public key of the joining node. If a joining node is granted membership of the S2 Access Control or S2 Authenticated class by a controller, the joining node advertises a ECDH public key where the first 16 bits have been set to zero. [136] If a joining node is only granted membership of the S2

Unauthenticated class, the node advertises the complete ECDH public key, so that the authentication step can be skipped.
Key exchange is done with Diffie-Hellman. S2 nodes use Elliptic Curve cryptography shared key to derive a temporary link key. S0 and S2 classes use AES-128 based network keys which are symmetric. S2 security class node can encrypt and decrypt commands using the same network key. Asymmetric ECDH keys used for the temporary secure channel are only used to send a few frames before nodes switch to the assigned S2 Network keys. [136]
S2 Network keys are physically stored in S2 nodes. The S2 transport layer integrates the S2 out-of-sync Nonce error messages with the supervision application-layer acknowledgements. If a frame can be decrypted, a supervision report is returned and if the frame cannot be decrypted, an S2 out-of-sync Nonce error message is returned. After Nonce resynchronization, the command may be re-transmitted. [136]
Z-Wave S2 bootstrapping message exchange has been shown in Figure 9.
Diffie-Hellman key exchange has been described briefly in Appendix 3

Figure 8: Z-Wave Authentication Sequence [6]

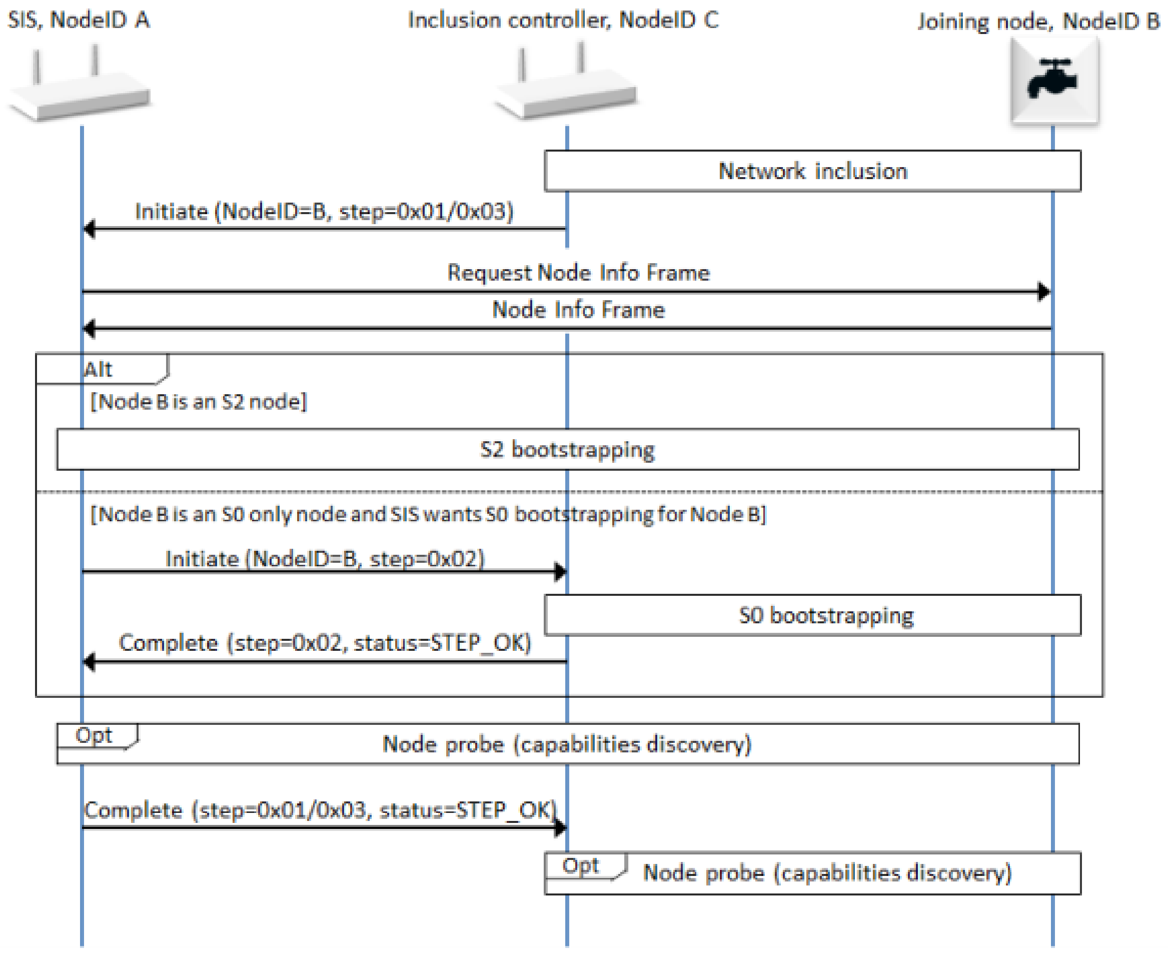
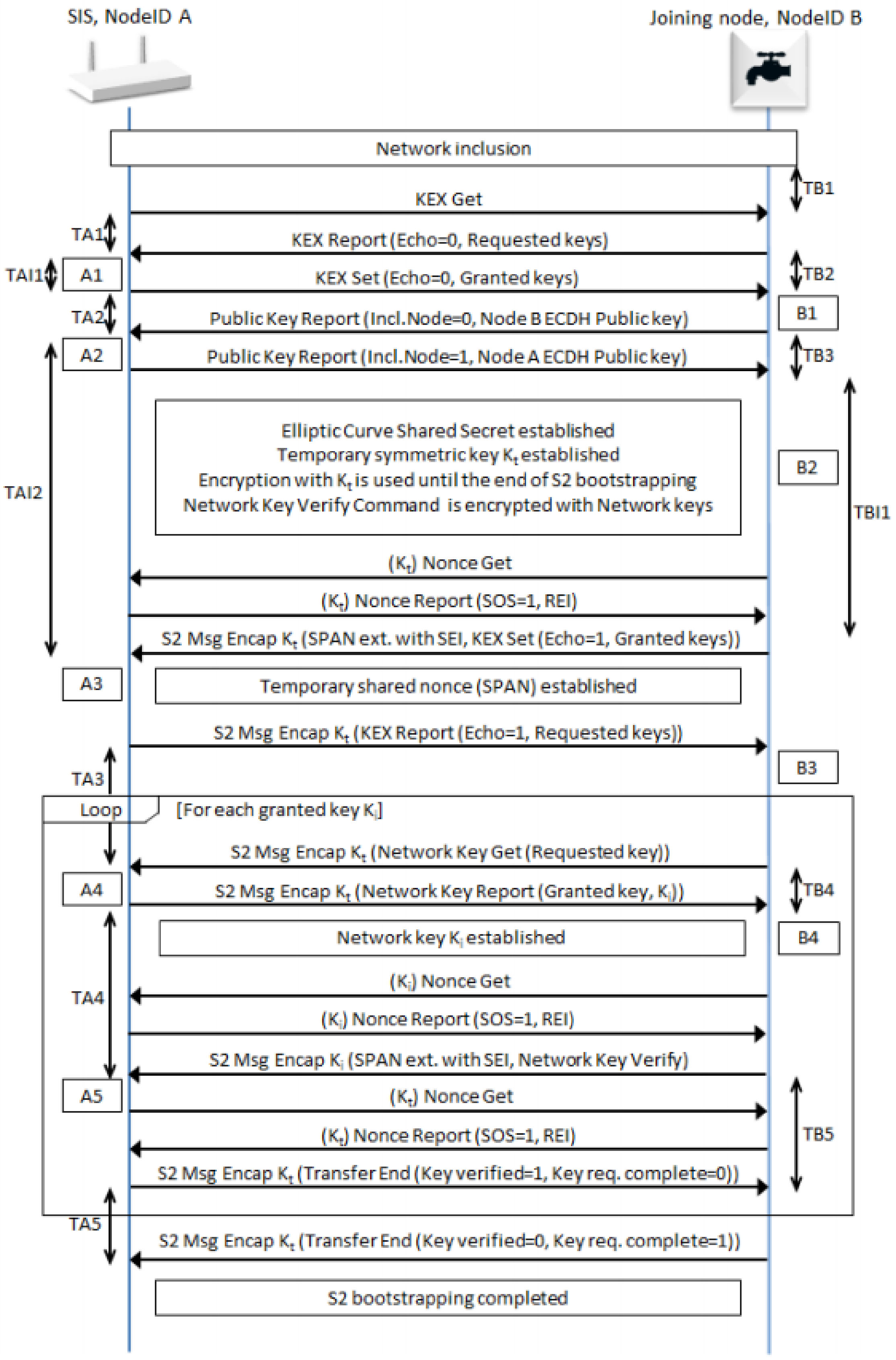

SIS, NodeID A
Inclusion controller, NodeID C
Joining node, NodeID B
Network inclusion
Initiate (NodeID=B, step=0x01/0x03)
Request Node Info Frame
Node Info Frame
Alt
[Node B is an S2 node]
S2 bootstrapping
[Node B is an S0 only node and SIS wants S0 bootstrapping for Node B]
Initiate (NodeID=B, step=0x02)
S0 bootstrapping
Complete (step=0x02, status=STEP_OK)
Opt
Node probe (capabilities discovery)
Complete (step=0x01/0x03, status=STEP_OK)
Opt
Node probe (capabilities discovery)

Figure 9: Z-Wave S2 Bootstrapping [6]


SIS, NodeID A
Joining node, NodeID B
Network inclusion
TB1
KEX Get
TA1
KEX Report (Echo=0, Requested keys)
TAI1
A1
KEX Set (Echo=0, Granted keys)
TB2
TA2
B1
Public Key Report (Incl.Node=0, Node B ECDH Public key)
A2
Public Key Report (Incl.Node=1, Node A ECDH Public key)
TB3
Elliptic Curve Shared Secret established
Temporary symmetric key K_t established
Encryption with K_t is used until the end of S2 bootstrapping
Network Key Verify Command is encrypted with Network keys
B2
TAI2
TBI1
(K_t) Nonce Get
(K_t) Nonce Report (SOS=1, REI)
S2 Msg Encap K_t (SPAN ext. with SEI, KEX Set (Echo=1, Granted keys))
A3
Temporary shared nonce (SPAN) established
S2 Msg Encap K_t (KEX Report (Echo=1, Requested keys))
B3
TA3
Loop
[For each granted key K_i]
S2 Msg Encap K_t (Network Key Get (Requested key))
A4
S2 Msg Encap K_t (Network Key Report (Granted key, K_i))
TB4
Network key K_i established
B4
TA4
(K_i) Nonce Get
(K_i) Nonce Report (SOS=1, REI)
S2 Msg Encap K_i (SPAN ext. with SEI, Network Key Verify)
A5
(K_t) Nonce Get
(K_t) Nonce Report (SOS=1, REI)
TB5
S2 Msg Encap K_t (Transfer End (Key verified=1, Key req. complete=0))
TA5
S2 Msg Encap K_t (Transfer End (Key verified=0, Key req. complete=1))
S2 bootstrapping completed

ProVerif Verification

```
set traceDisplay = long.

query attacker (SharedNetworkKey).
query attacker (DEC_SharedNetworkKey).
query attacker (RNonce).
query attacker (DEC_RealNonce).
query attacker (SharedNetworkKey_temp).
query attacker (RNonce_temp).
query attacker (DEC_Finish).
query attacker (Finish).

query x:bitstring; event(StartConnection(x)) ==> event(AcceptConnection(x)
   ).
query x:bitstring; inj-event(StartConnection(x)) ==> inj-event(
   AcceptConnection(x)).


type key.
type pin.
type nonce.
type realnonce.
type realkey.

free c:channel.

free RealNonceRequest:bitstring [private].
free NetworkInformation_SIS:bitstring [private].
free NetworkInofrmation_JoiningNode:bitstring [private].
free KEX_Request:bitstring [private].
free KEX_Response:bitstring [private].
free KEX_SET:bitstring [private].
free PubKeyJoiningNode:key [private].
free UserSISPin:bitstring [private].
free PubKeySIS:key [private].
free JoiningDSK:pin [private].
free JoiningNodeTempSymKey:key[private].
free SISTempSymKey:key[private].
free TempNonce:nonce[private].
free NonceRequest:bitstring [private].
free DEC_KEX_SET:nonce [private].
free ENC_Sec2KeyGet:nonce[private].
free DEC_Sec2KeyGet:bitstring[private].
free SharedNetworkKey:realkey[private].
free ENC_SharedNetworkKey:nonce[private].
free DEC_SharedNetworkKey:realkey[private].
free ENC_RealNonceReq:nonce[private].
free DEC_RealNonceReq:bitstring [private].
free ENC_RealNonce:nonce [private].
free DEC_RealNonce:realnonce [private].
free RNonce:realnonce [private].
free Verification:bitstring [private].
free ENC_Verification:nonce [private].
free DEC_Verification:bitstring [private].
free KeyVerified:bitstring [private].
free ENCKeyVerified:nonce [private].
free DEC_KeyVerified:bitstring [private].
free Finish:bitstring [private].
free ENCFinish:nonce [private].
free DEC_Finish:bitstring [private].
free Sec2KeyGet:bitstring [private].
free SharedNetworkKey_temp:bitstring [private].
free RNonce_temp:bitstring [private].

event KEX_SET_Verification(bitstring).
event KEX_SET_Decryption(bitstring).
event KEX_Req_Decryption(bitstring).
event KEX_Response_Verification(bitstring).
event Sec2KeyGet_Decryption(bitstring).
event RealNonce_Decryption(realnonce).
event Verification_Decryption(bitstring).
event KeyVerifiedDecryption(bitstring).
event FinishDecryption(bitstring).
event RealNetworkKey(realkey).
event RealNonceRequest_Decryption(bitstring).
event SharedNetworkKeyGeneration(realkey).
event RNonceGeneration(realnonce).
event StartConnection(bitstring).
event AcceptConnection(bitstring).

fun TempSymENC(bitstring, nonce, key): nonce.
reduc forall x: bitstring, y: nonce, z:key; TempSymDEC(TempSymENC(x,y,z),
   z) = x.
```

```
fun SecureComm(bitstring, realnonce, realkey):nonce.
reduc forall x:bitstring, y:realnonce, z:realkey; DecSecureComm(SecureComm
   (x,y,z),z) = x.

fun tc1(bitstring) : realnonce [typeConverter].
fun tc2(bitstring) : realkey [typeConverter].


let JoiningNode =
  out(c, NetworkInofrmation_JoiningNode).

let SISNode =
  in(c, NetworkInformation_SIS:bitstring);
  out(c, NetworkInformation_SIS).

let JoiningNode2 =
  in(c, NetworkInformation_SIS:bitstring).

let SISNode2 =
  out(c, KEX_Request).

let JoiningNode3 =
  in(c, KEX_Request:bitstring);
  out(c, KEX_Response).

let SISNode3 =
  in(c, KEX_Response:bitstring);
  let NetworkInformation_SIS = KEX_Response in
    event KEX_Response_Verification(KEX_Response);
  if NetworkInformation_SIS = KEX_Response then
  (
      out(c, KEX_SET)
    )
    else
    (
      0
      ).

let JoiningNode4 =
  in(c, KEX_SET:bitstring);
  let KEX_SET = KEX_Request in
```

```
    event KEX_SET_Verification(KEX_SET);
  if KEX_SET = KEX_Request then
  (
    out(c, PubKeyJoiningNode)
    )
    else
    (
      0
      ).

  let SISNode4 =
    in(c, PubKeyJoiningNode:key);
    in(c, UserSISPin:bitstring);
    if UserSISPin = UserSISPin then
    (

      out(c, PubKeySIS)
      )
      else
      (
        0
        ).

  let JoiningNode5 =
    in(c, PubKeySIS:key);
    in(c, JoiningDSK:pin);
    if JoiningDSK = JoiningDSK then
    (

      out(c, NonceRequest)
      )
      else
      (
        0
        ).
let SISNode5 =
  in(c, NonceRequest: bitstring);
  out(c, TempNonce).

let JoiningNode6=
  in(c, TempNonce:nonce);
  out(c, TempSymENC(KEX_SET,TempNonce,JoiningNodeTempSymKey)).
```

```
let SISNode6=
  in(c, ENC_KEX_SET:nonce);
  let DEC_KEX_SET = TempSymDEC(ENC_KEX_SET, SISTempSymKey) in
    event KEX_SET_Decryption(DEC_KEX_SET);
  if DEC_KEX_SET = KEX_SET then
  (
    out(c, TempSymENC(KEX_Request,TempNonce,SISTempSymKey))
    )
    else
    (
      0
      ).

let JoiningNode7 =
  in(c, ENC_KEX_Req:nonce);
  let DEC_KEX_Req = TempSymDEC(ENC_KEX_Req, JoiningNodeTempSymKey) in
    event KEX_Req_Decryption(DEC_KEX_Req);
  if DEC_KEX_Req = KEX_Request then
  (
    out(c, TempSymENC(Sec2KeyGet, TempNonce, JoiningNodeTempSymKey))
    )
    else
    (
      0
      ).

  let SISNode7 =
    in(c, ENC_Sec2KeyGet:nonce);
    let DEC_Sec2KeyGet = TempSymDEC(ENC_Sec2KeyGet, SISTempSymKey) in
      event Sec2KeyGet_Decryption(DEC_Sec2KeyGet);
    let SharedNetworkKey = tc2(SharedNetworkKey_temp) in
      event SharedNetworkKeyGeneration(SharedNetworkKey);
    out(c, TempSymENC(SharedNetworkKey_temp,TempNonce,SISTempSymKey)).

  let JoiningNode8 =
    in(c, ENC_SharedNetworkKey:nonce);
    let DEC_SharedNetworkKey = TempSymDEC(ENC_SharedNetworkKey,
        JoiningNodeTempSymKey) in
      event RealNetworkKey(tc2(DEC_SharedNetworkKey));
    out(c, TempSymENC(RealNonceRequest, TempNonce, JoiningNodeTempSymKey))
        .
```

```
let SISNode8 =
  in(c, ENC_RealNonceReq:nonce);
  let DEC_RealNonceReq = TempSymDEC(ENC_RealNonceReq, SISTempSymKey) in
    event RealNonceRequest_Decryption(DEC_RealNonceReq);
  let RNonce = tc1(RNonce_temp) in
    event RNonceGeneration(RNonce);
  out(c,TempSymENC(RNonce_temp, TempNonce, SISTempSymKey)).

let JoiningNode9 =
  in(c, ENC_RealNonce:nonce);
  let DEC_RealNonce = TempSymDEC(ENC_RealNonce, JoiningNodeTempSymKey)
      in
    event RealNonce_Decryption(tc1(DEC_RealNonce));
  out(c, SecureComm(Verification, tc1(DEC_RealNonce),
      DEC_SharedNetworkKey)).

let SISNode9 =
  in (c, ENC_Verification:nonce);
  let DEC_Verification = DecSecureComm(ENC_Verification,
      SharedNetworkKey) in
    event Verification_Decryption(DEC_Verification);
  if DEC_Verification = Verification then
  (
    out(c, SecureComm(KeyVerified, RNonce, SharedNetworkKey))
    )
    else
    (
      0
      ).

let JoiningNode10 =
  in(c, ENCKeyVerified:nonce);
  let DEC_KeyVerified = DecSecureComm(ENCKeyVerified,
      DEC_SharedNetworkKey) in
    event KeyVerifiedDecryption(DEC_KeyVerified);
  if DEC_KeyVerified = KeyVerified then
  (
    event StartConnection(Finish);
    out(c, SecureComm(Finish, DEC_RealNonce, DEC_SharedNetworkKey))
    )
    else
```

```
        (
          0
          ).

    let SISNode10 =
      in(c, ENCFinish:nonce);
      let DEC_Finish = DecSecureComm(ENCFinish, SharedNetworkKey) in
        event FinishDecryption(DEC_Finish);
      if DEC_Finish = Finish then
      (
        event AcceptConnection(DEC_Finish)
        )
        else
        (
          0
          ).
process
    (
      !JoiningNode|!SISNode|!JoiningNode2|!SISNode2|!JoiningNode3|!SISNode3
         |!JoiningNode4|!SISNode4|!JoiningNode5|!SISNode5|!JoiningNode6|!
         SISNode6|!JoiningNode7|!SISNode7|!JoiningNode8|!SISNode8|!
         JoiningNode9|!SISNode9|!JoiningNode10|!SISNode10
      )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, NetworkInofrmation_JoiningNode)
) | (
    {3}!
    {4}in(c, NetworkInformation_SIS_25: bitstring);
    {5}out(c, NetworkInformation_SIS_25)
) | (
    {6}!
    {7}in(c, NetworkInformation_SIS_26: bitstring)
) | (
    {8}!
    {9}out(c, KEX_Request)
) | (
    {10}!
    {11}in(c, KEX_Request_27: bitstring);
    {12}out(c, KEX_Response)
) | (
    {13}!
    {14}in(c, KEX_Response_28: bitstring);
    {15}let NetworkInformation_SIS_29: bitstring = KEX_Response_28 in
    {16}event KEX_Response_Verification(KEX_Response_28);
    {17}if (NetworkInformation_SIS_29 = KEX_Response_28) then
    {18}out(c, KEX_SET)
) | (
    {19}!
    {20}in(c, KEX_SET_30: bitstring);
    {21}let KEX_SET_31: bitstring = KEX_Request in
    {22}event KEX_SET_Verification(KEX_SET_31);
    {23}if (KEX_SET_31 = KEX_Request) then
    {24}out(c, PubKeyJoiningNode)
) | (
    {25}!
    {26}in(c, PubKeyJoiningNode_32: key);
    {27}in(c, UserSISPin_33: bitstring);
    {28}if (UserSISPin_33 = UserSISPin_33) then
    {29}out(c, PubKeySIS)
) | (
    {30}!
    {31}in(c, PubKeySIS_34: key);
    {32}in(c, JoiningDSK_35: pin);
    {33}if (JoiningDSK_35 = JoiningDSK_35) then
    {34}out(c, NonceRequest)
) | (
    {35}!
    {36}in(c, NonceRequest_36: bitstring);
    {37}out(c, TempNonce)
) | (
    {38}!
    {39}in(c, TempNonce_37: nonce);
    {40}out(c, TempSymENC(KEX_SET,TempNonce_37,JoiningNodeTempSymKey))
) | (
    {41}!
    {42}in(c, ENC_KEX_SET: nonce);
    {43}let DEC_KEX_SET_38: bitstring = TempSymDEC(ENC_KEX_SET,
       SISTempSymKey) in
    {44}event KEX_SET_Decryption(DEC_KEX_SET_38);
    {45}if (DEC_KEX_SET_38 = KEX_SET) then
    {46}out(c, TempSymENC(KEX_Request,TempNonce,SISTempSymKey))
) | (
    {47}!
    {48}in(c, ENC_KEX_Req: nonce);
    {49}let DEC_KEX_Req: bitstring = TempSymDEC(ENC_KEX_Req,
       JoiningNodeTempSymKey) in
    {50}event KEX_Req_Decryption(DEC_KEX_Req);
    {51}if (DEC_KEX_Req = KEX_Request) then
    {52}out(c, TempSymENC(Sec2KeyGet,TempNonce,JoiningNodeTempSymKey))
) | (
    {53}!
    {54}in(c, ENC_Sec2KeyGet_39: nonce);
    {55}let DEC_Sec2KeyGet_40: bitstring = TempSymDEC(ENC_Sec2KeyGet_39,
       SISTempSymKey) in
    {56}event Sec2KeyGet_Decryption(DEC_Sec2KeyGet_40);
    {57}let SharedNetworkKey_41: realkey = SharedNetworkKey_temp in
    {58}event SharedNetworkKeyGeneration(SharedNetworkKey_41);
    {59}out(c, TempSymENC(SharedNetworkKey_temp,TempNonce,SISTempSymKey))
) | (
    {60}!
    {61}in(c, ENC_SharedNetworkKey_42: nonce);
    {62}let DEC_SharedNetworkKey_43: bitstring = TempSymDEC(
       ENC_SharedNetworkKey_42,JoiningNodeTempSymKey) in
    {63}event RealNetworkKey(DEC_SharedNetworkKey_43);
    {64}out(c, TempSymENC(RealNonceRequest,TempNonce,JoiningNodeTempSymKey
       ))
) | (
    {65}!
    {66}in(c, ENC_RealNonceReq_44: nonce);
    {67}let DEC_RealNonceReq_45: bitstring = TempSymDEC(
       ENC_RealNonceReq_44,SISTempSymKey) in
    {68}event RealNonceRequest_Decryption(DEC_RealNonceReq_45);
    {69}let RNonce_46: realnonce = RNonce_temp in
    {70}event RNonceGeneration(RNonce_46);
    {71}out(c, TempSymENC(RNonce_temp,TempNonce,SISTempSymKey))
) | (
    {72}!
    {73}in(c, ENC_RealNonce_47: nonce);
    {74}let DEC_RealNonce_48: bitstring = TempSymDEC(ENC_RealNonce_47,
       JoiningNodeTempSymKey) in
    {75}event RealNonce_Decryption(DEC_RealNonce_48);
    {76}out(c, SecureComm(Verification,DEC_RealNonce_48,
       DEC_SharedNetworkKey))
) | (
    {77}!
    {78}in(c, ENC_Verification_49: nonce);
    {79}let DEC_Verification_50: bitstring = DecSecureComm(
       ENC_Verification_49,SharedNetworkKey) in
    {80}event Verification_Decryption(DEC_Verification_50);
    {81}if (DEC_Verification_50 = Verification) then
    {82}out(c, SecureComm(KeyVerified,RNonce,SharedNetworkKey))
) | (
    {83}!
    {84}in(c, ENCKeyVerified_51: nonce);
    {85}let DEC_KeyVerified_52: bitstring = DecSecureComm(
       ENCKeyVerified_51,DEC_SharedNetworkKey) in
    {86}event KeyVerifiedDecryption(DEC_KeyVerified_52);
    {87}if (DEC_KeyVerified_52 = KeyVerified) then
    {88}event StartConnection(Finish);
    {89}out(c, SecureComm(Finish,DEC_RealNonce,DEC_SharedNetworkKey))
) | (
    {90}!
    {91}in(c, ENCFinish_53: nonce);
    {92}let DEC_Finish_54: bitstring = DecSecureComm(ENCFinish_53,
       SharedNetworkKey) in
    {93}event FinishDecryption(DEC_Finish_54);
    {94}if (DEC_Finish_54 = Finish) then
    {95}event AcceptConnection(DEC_Finish_54)
)

-- Query not attacker(SharedNetworkKey[])
Completing...
Starting query not attacker(SharedNetworkKey[])
RESULT not attacker(SharedNetworkKey[]) is true.
-- Query not attacker(DEC_SharedNetworkKey[])
Completing...
Starting query not attacker(DEC_SharedNetworkKey[])
RESULT not attacker(DEC_SharedNetworkKey[]) is true.
-- Query not attacker(RNonce[])
Completing...
Starting query not attacker(RNonce[])
RESULT not attacker(RNonce[]) is true.
-- Query not attacker(DEC_RealNonce[])
Completing...
Starting query not attacker(DEC_RealNonce[])
RESULT not attacker(DEC_RealNonce[]) is true.
-- Query not attacker(SharedNetworkKey_temp[])
Completing...
Starting query not attacker(SharedNetworkKey_temp[])
RESULT not attacker(SharedNetworkKey_temp[]) is true.
-- Query not attacker(RNonce_temp[])
Completing...
Starting query not attacker(RNonce_temp[])
RESULT not attacker(RNonce_temp[]) is true.
-- Query not attacker(DEC_Finish[])
Completing...
Starting query not attacker(DEC_Finish[])
RESULT not attacker(DEC_Finish[]) is true.
-- Query not attacker(Finish[])
Completing...
Starting query not attacker(Finish[])
RESULT not attacker(Finish[]) is true.
-- Query event(StartConnection(x_55)) ==> event(AcceptConnection(x_55))
Completing...
Starting query event(StartConnection(x_55)) ==> event(AcceptConnection(
   x_55))
RESULT event(StartConnection(x_55)) ==> event(AcceptConnection(x_55)) is
```

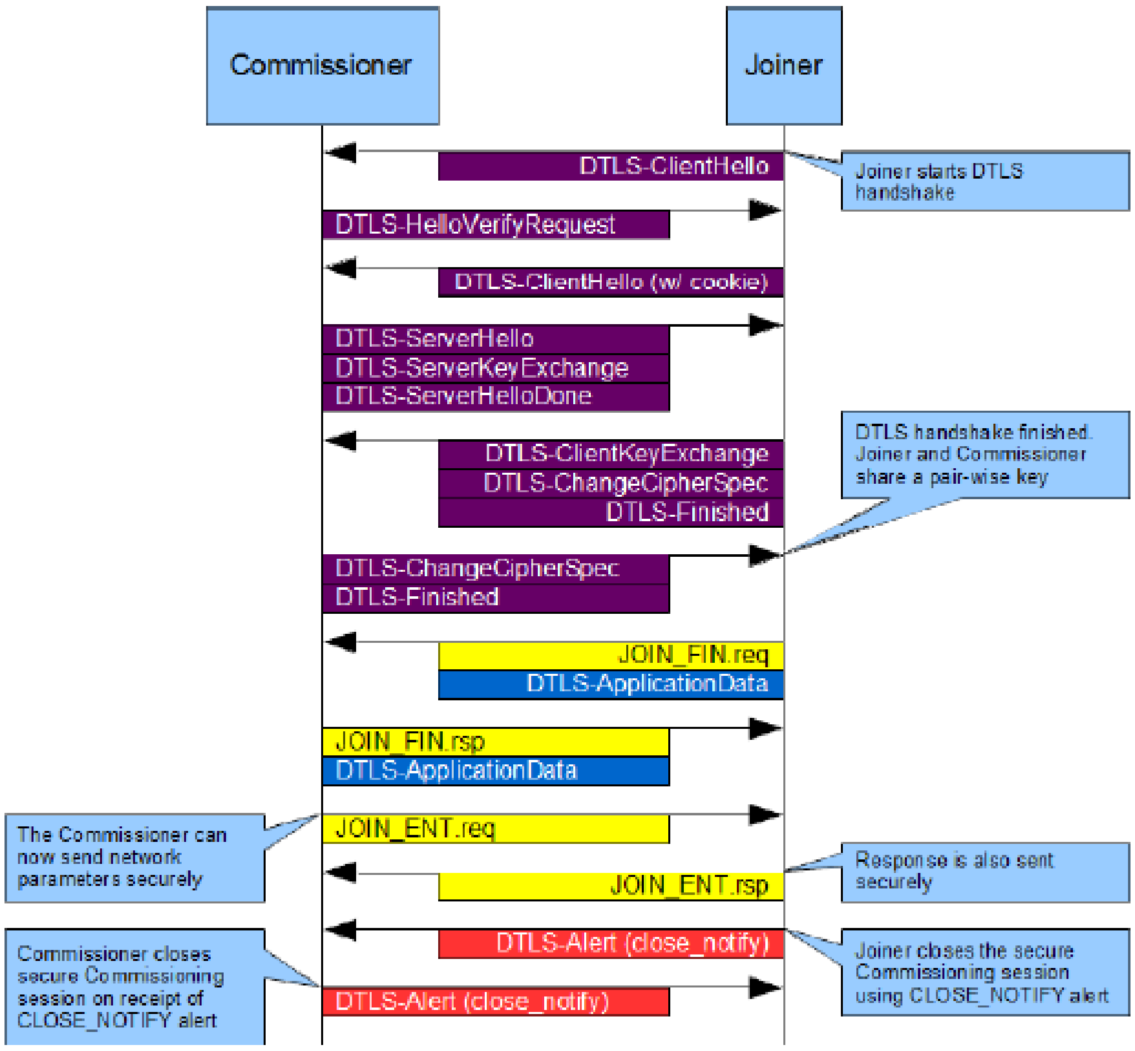
```
    true.
-- Query inj-event(StartConnection(x_56)) ==> inj-event(AcceptConnection(
    x_56))
Completing...
Starting query inj-event(StartConnection(x_56)) ==> inj-event(
    AcceptConnection(x_56))
RESULT inj-event(StartConnection(x_56)) ==> inj-event(AcceptConnection(
    x_56)) is true.
```

Appendix 7 – Thread Authentication

Commissioning or authenticating must be able to take place in a system where a Joiner that wishes to join to the Thread Network is authenticated using a device known as a Commissioner, Authentication Server, which must be connected to the already existing Thread network to authenticate. Commissioning can take place from being connected to different network, if there is appropriate route to the Thread Network.
Thread uses elliptic curve variant of J-PAKE (EC-JPAKE), using the NIST P-256 elliptic curve. J-PAKE [145] is a password-authenticated key exchange (PAKE) with juggling. It essentially uses elliptic curve Diffie-Hellman for key agreement and Schnorr signatures as a NIZK (Non-Interactive Zero-Knowledge) [146] proof mechanism to authenticate two peers and to establish a shared secret between them based on the passphrase. [7]
Thread network is protected with a network-wide key, which is used at the MAC (Media Access Control) layer to protect the MAC data frames. As it is a network key, compromise of any Thread device could potentially reveal the key; therefore, it is not typically used as the only form of protection within the Thread Network. [7]
When joining Thread Network, device is required to identify router which is used to join and to communicate in a point-to-point protocol way. Router polices any traffic from the device and forwards it to the authentication server in a controlled manner to allow the authentication protocol (DTLS handshake) to execute. DTLS key exchange is described in Appendix 10.
If the Commissioner is not in direct communication with the Joiner, the Joiner Router must relay the DTLS handshake with the Commissioner. The Commissioning relay protocol encapsulates DTLS handshake and relays of the DTLS handshake from the Joiner all the way to the Commissioner. Authentication server uses Commissioning protocol to keep a secure communication session, it is based on CoAP [104].
There are two different Commissioner appointing set ups, external Commissioner and native.
If the Commissioner candidate uses a WLAN network for commissioning purposes, it is known as an external Commissioner. An external Commissioner has to connect the Thread network through a Thread router, which is known as border router to become authorized Commissioner. Commissioner candidate must use an authentication handshake with a router to prove it is eligible to become authorized Commissioner and set up a secure Commissioning session. The Commissioner candidate then connects to the Leader through the border router because there can be only one authorized Commissioner. If

connection and validation succeeds, the Commissioner candidate becomes authorized external Commissioner. The secure Commissioning session remains in place and the representative border router will be made known of the result in the Thread network, because all subsequent communication with other Thread devices will be done through that border router. [7]
If the Commissioner candidate uses a Thread network interface for commissioning, it is known as a Native Commissioner. A Native Commissioner has to petition the Thread Network through a representative Commissioner router) to become authorized Commissioner. The Commissioner candidate must use an authentication handshake with the Commissioner router to prove it is eligible to become authorized Commissioner and set up a secure Commissioning session. The Commissioner Candidate then connects with the Leader via the Commissioner router because there can be only one authorized Commissioner. If validation succeeds, the Commissioner candidate becomes the sole authorized Native Commissioner. However, the Commissioner subsequently joins the Thread Network and becomes an active device (on-mesh Commissioner) and all communication with other Thread devices takes place directly with the Commissioner. [7]
In this method, evaluation is done based on Joiner–Joiner Router/Commissioner sequence, however all these sequences are similar, with the difference of the which Commissioner or router is used. This message exchange has been show in Figure 10.
This communication is over an unsecured radio link and all traffic between the Joiner and Joiner Router will be sent in the clear without any form of integrity checking. This fundamentally means the Joiner Router has to treat any traffic from the Joiner as completely unauthenticated. Normally, the Thread Network would be in a “lock down” mode, which would cause any Thread Device on the perimeter to ignore any unsecured 802.15.4 traffic. However, when joining is permitted, Joiner Routers should carefully police unsecured 802.15.4 traffic and assume it to be authentication traffic. [7]
The DTLS handshake will occur as initial communication being established between the Joiner and Joiner Router. A relay agent will check the incoming traffic and the Joiner Router will relay the DTLS client handshake along with address and port details of the Joiner and the Joiner Router itself to the Border Router. The address and port details ensure relayed DTLS server handshake response messages can be relayed back through the Joiner Router to the originating Joiner. [7]
As mentioned earlier, DTLS key exchange is described in Appendix 10.

Figure 10: Thread Joiner–Joiner Router/Commissioner Sequence [7]


Commissioner
Joiner
DTLS-ClientHello
Joiner starts DTLS handshake
DTLS-HelloVerifyRequest
DTLS-ClientHello (w/ cookie)
DTLS-ServerHello
DTLS-ServerKeyExchange
DTLS-ServerHelloDone
DTLS handshake finished. Joiner and Commissioner share a pair-wise key
DTLS-ClientKeyExchange
DTLS-ChangeCipherSpec
DTLS-Finished
DTLS-ChangeCipherSpec
DTLS-Finished
JOIN_FIN.req
DTLS-ApplicationData
JOIN_FIN.rsp
DTLS-ApplicationData
The Commissioner can now send network parameters securely
JOIN_ENT.req
JOIN_ENT.rsp
Response is also sent securely
DTLS-Alert (close_notify)
Joiner closes the secure Commissioning session using CLOSE_NOTIFY alert
Commissioner closes secure Commissioning session on receipt of CLOSE_NOTIFY alert
DTLS-Alert (close_notify)

Thread Joiner-Commissioner ProVerif Verification

```
set traceDisplay = long.

query attacker(CommissionerPrivKey).
query attacker(JoinerPrivKey).
query attacker(JoinerMKey).
query attacker(CommissionerMKey).
query attacker(JoinerDecryptedData1).
query attacker(CommissionerDecryptedData1).

query x:bitstring; event(JoinerDecryptData1(x)).
query x:bitstring; event(CommissionerDecryptData1(x)).
query x:bitstring; event(JoinerDecryptData1(x)) ==> event(
   CommissionerDecryptData1(x)).
query x:bitstring; inj-event(JoinerDecryptData1(x)) ==> inj-event(
   CommissionerDecryptData1(x)).

type key.
type nonce.
type mkey.

free c:channel.

free AppData:bitstring[private].
free JoinedReq:bitstring[private].
free JoinFinishedVerified:bitstring[private].
free JoinedFinished:bitstring[private].
free Close, HelloCookie, JoinFinished, RealHello:bitstring[private].
free Close_ACK, JoinerDecryptedData1:bitstring[private].
free CommissionerDecryptedData1:bitstring[private].
free CommissionerDecryptedData2:bitstring[private].
free JoinerDecData1:bitstring[private].
free JoinerDecData2, Hello, HelloAck:bitstring[private].

free CommissionerPubKey:key [private].
free CommissionerPrivKey:key [private].
free JoinerPubKey:key [private].
free JoinerPrivKey:key [private].

free CommissionerMKey:mkey [private].
free JoinerMKey:mkey [private].
```

```
fun CommunicationKeyGeneration(bitstring, key, key):mkey.
event JoinerMKeyGeneration(mkey).
event CommissionerMKeyGeneration(mkey).

event JoinerDecryptData1(bitstring).
event JoinerDecryptData2(bitstring).
event CommissionerDecryptData1(bitstring).
event CommissionerDecryptData2(bitstring).

fun Encryption(bitstring, mkey): nonce.
reduc forall x: bitstring, y: mkey; Decryption(Encryption(x,y), y) = x.


let Joiner =
  new Hello:bitstring;
  out(c, Hello).

let Commissioner =
  in(c, Hello:bitstring);
  new HelloVerification: bitstring;
  out(c, HelloVerification).

let Joiner2 =
  in(c, HelloVerification:bitstring);
  new HelloCookie:bitstring;
  out(c, HelloCookie).

let Commissioner2 =
  in(c, HelloCookie:bitstring);
  new CommissionerPubKey:key;
  out(c, (RealHello, CommissionerPubKey)).

let Joiner3 =
  in(c, (RealHello:bitstring, CommissionerPubKey:key));
  new JoinerPubKey:key;
  new RealHelloFinished:bitstring;
  out(c, (RealHelloFinished, JoinerPubKey)).

let Commissioner3 =
  in(c, (RealHelloFinished:bitstring, JoinerPubKey:key));
  new Finished:bitstring;
  out(c, Finished).
```

```
let Joiner4 =
  in(c, Finished:bitstring);
  new JoinerPrivKey:key;
  let JoinerMKey = CommunicationKeyGeneration(RealHello,
      CommissionerPubKey, JoinerPrivKey) in
    event JoinerMKeyGeneration(JoinerMKey);
  out(c, Encryption((AppData, JoinFinished),JoinerMKey)).

let Commissioner4 =
  in (c, JoinerEncData1:nonce);
  new CommissionerPrivKey:key;
  let CommissionerMKey = CommunicationKeyGeneration(HelloCookie,
      JoinerPubKey, CommissionerPrivKey) in
    event CommissionerMKeyGeneration(CommissionerMKey);
  let JoinerDecryptedData1 = Decryption(JoinerEncData1, CommissionerMKey)
      in
    event JoinerDecryptData1(JoinerDecryptedData1);
  out(c, Encryption((AppData, JoinFinishedVerified), CommissionerMKey)).

let Joiner5 =
  in(c, CommissionerEncData1:nonce);
  new JoinerPrivKey:key;
  let CommissionerDecryptedData1 = Decryption(CommissionerEncData1,
      JoinerMKey) in
    event CommissionerDecryptData1(CommissionerDecryptedData1).

process (
  !Joiner|!Commissioner|!Joiner2|!Commissioner2|!Joiner3|!Commissioner3|!
      Joiner4|!Commissioner4|!Joiner5
  )
```

Thread Joiner-Commissioner ProVerif Result

```
Process:
(
    {1}!
    {2}new Hello_13: bitstring;
    {3}out(c, Hello_13)
) | (
    {4}!
    {5}in(c, Hello_14: bitstring);
    {6}new HelloVerification: bitstring;
    {7}out(c, HelloVerification)
) | (
    {8}!
    {9}in(c, HelloVerification_15: bitstring);
    {10}new HelloCookie_16: bitstring;
    {11}out(c, HelloCookie_16)
) | (
    {12}!
    {13}in(c, HelloCookie_17: bitstring);
    {14}new CommissionerPubKey_18: key;
    {15}out(c, (RealHello,CommissionerPubKey_18))
) | (
    {16}!
    {17}in(c, (RealHello_19: bitstring,CommissionerPubKey_20: key));
    {18}new JoinerPubKey_21: key;
    {19}new RealHelloFinished: bitstring;
    {20}out(c, (RealHelloFinished,JoinerPubKey_21))
) | (
    {21}!
    {22}in(c, (RealHelloFinished_22: bitstring,JoinerPubKey_23: key));
    {23}new Finished: bitstring;
    {24}out(c, Finished)
) | (
    {25}!
    {26}in(c, Finished_24: bitstring);
    {27}new JoinerPrivKey_25: key;
    {28}let JoinerMKey_26: mkey = CommunicationKeyGeneration(RealHello,
       CommissionerPubKey,JoinerPrivKey_25) in
    {29}event JoinerMKeyGeneration(JoinerMKey_26);
    {30}out(c, Encryption((AppData,JoinFinished),JoinerMKey_26))
) | (
    {31}!
    {32}in(c, JoinerEncData1: nonce);
    {33}new CommissionerPrivKey_27: key;
    {34}let CommissionerMKey_28: mkey = CommunicationKeyGeneration(
       HelloCookie,JoinerPubKey,CommissionerPrivKey_27) in
    {35}event CommissionerMKeyGeneration(CommissionerMKey_28);
    {36}let JoinerDecryptedData1_29: bitstring = Decryption(JoinerEncData1
       ,CommissionerMKey_28) in
    {37}event JoinerDecryptData1(JoinerDecryptedData1_29);
    {38}out(c, Encryption((AppData,JoinFinishedVerified),
       CommissionerMKey_28))
) | (
    {39}!
    {40}in(c, CommissionerEncData1: nonce);
    {41}new JoinerPrivKey_30: key;
    {42}let CommissionerDecryptedData1_31: bitstring = Decryption(
       CommissionerEncData1,JoinerMKey) in
    {43}event CommissionerDecryptData1(CommissionerDecryptedData1_31)
)

-- Query not attacker(CommissionerPrivKey[])
Completing...
Starting query not attacker(CommissionerPrivKey[])
RESULT not attacker(CommissionerPrivKey[]) is true.
-- Query not attacker(JoinerPrivKey[])
Completing...
Starting query not attacker(JoinerPrivKey[])
RESULT not attacker(JoinerPrivKey[]) is true.
-- Query not attacker(JoinerMKey[])
Completing...
Starting query not attacker(JoinerMKey[])
RESULT not attacker(JoinerMKey[]) is true.
-- Query not attacker(CommissionerMKey[])
Completing...
Starting query not attacker(CommissionerMKey[])
RESULT not attacker(CommissionerMKey[]) is true.
-- Query not attacker(JoinerDecryptedData1[])
Completing...
Starting query not attacker(JoinerDecryptedData1[])
RESULT not attacker(JoinerDecryptedData1[]) is true.
-- Query not attacker(CommissionerDecryptedData1[])
Completing...
```

```
Starting query not attacker(CommissionerDecryptedData1[])
RESULT not attacker(CommissionerDecryptedData1[]) is true.
-- Query not event(JoinerDecryptData1(x_32))
Completing...
Starting query not event(JoinerDecryptData1(x_32))
RESULT not event(JoinerDecryptData1(x_32)) is true.
-- Query not event(CommissionerDecryptData1(x_33))
Completing...
Starting query not event(CommissionerDecryptData1(x_33))
RESULT not event(CommissionerDecryptData1(x_33)) is true.
-- Query event(JoinerDecryptData1(x_34)) ==> event(
   CommissionerDecryptData1(x_34))
Completing...
Starting query event(JoinerDecryptData1(x_34)) ==> event(
   CommissionerDecryptData1(x_34))
RESULT event(JoinerDecryptData1(x_34)) ==> event(CommissionerDecryptData1(
   x_34)) is true.
-- Query inj-event(JoinerDecryptData1(x_35)) ==> inj-event(
   CommissionerDecryptData1(x_35))
Completing...
Starting query inj-event(JoinerDecryptData1(x_35)) ==> inj-event(
   CommissionerDecryptData1(x_35))
RESULT inj-event(JoinerDecryptData1(x_35)) ==> inj-event(
   CommissionerDecryptData1(x_35)) is true.
```

Appendix 8 – TLS Authentication

The TLS protocol enable two parties to identify and authenticate one another and communicate with confidentiality and data integrity. TLS connection is started by an application, which becomes TLS client. Every new connection begin with TLS handshake. Handshake enables the the client and server to establish the secret keys with which they communicate, from that point forward. TLS handshake means that client and server agree on the version of the protocol to use, select cryptographic algorithms, authenticate each other by exchanging and validating digital certificates and use asymmetric encryption techniques to generate a shared secret key. SSL or TLS then uses the shared key for the symmetric encryption of messages, which is faster than asymmetric encryption. TLS/SSL authentication message exchange has been show in Figure 11 and short description of it [8]:

1. The SSL or TLS client sends a client hello message that lists cryptographic information such as the SSL or TLS version and, in the client's order of preference, the cipher suites supported by the client. The message also contains a random byte string that is used in subsequent computations. The protocol allows for the client hello to include the data compression methods supported by the client.

2. The SSL or TLS server responds with a server hello message that contains the cipher suite chosen by the server from the list provided by the client, the session ID, and another random byte string. The server also sends its digital certificate. If the server requires a digital certificate for client authentication, the server sends a client certificate request that includes a list of the types of certificates supported and the Distinguished Names of acceptable Certification Authorities (CA-s).

3. The SSL or TLS client verifies the server's digital certificate. For more information, see How SSL and TLS provide identification, authentication, confidentiality, and integrity.

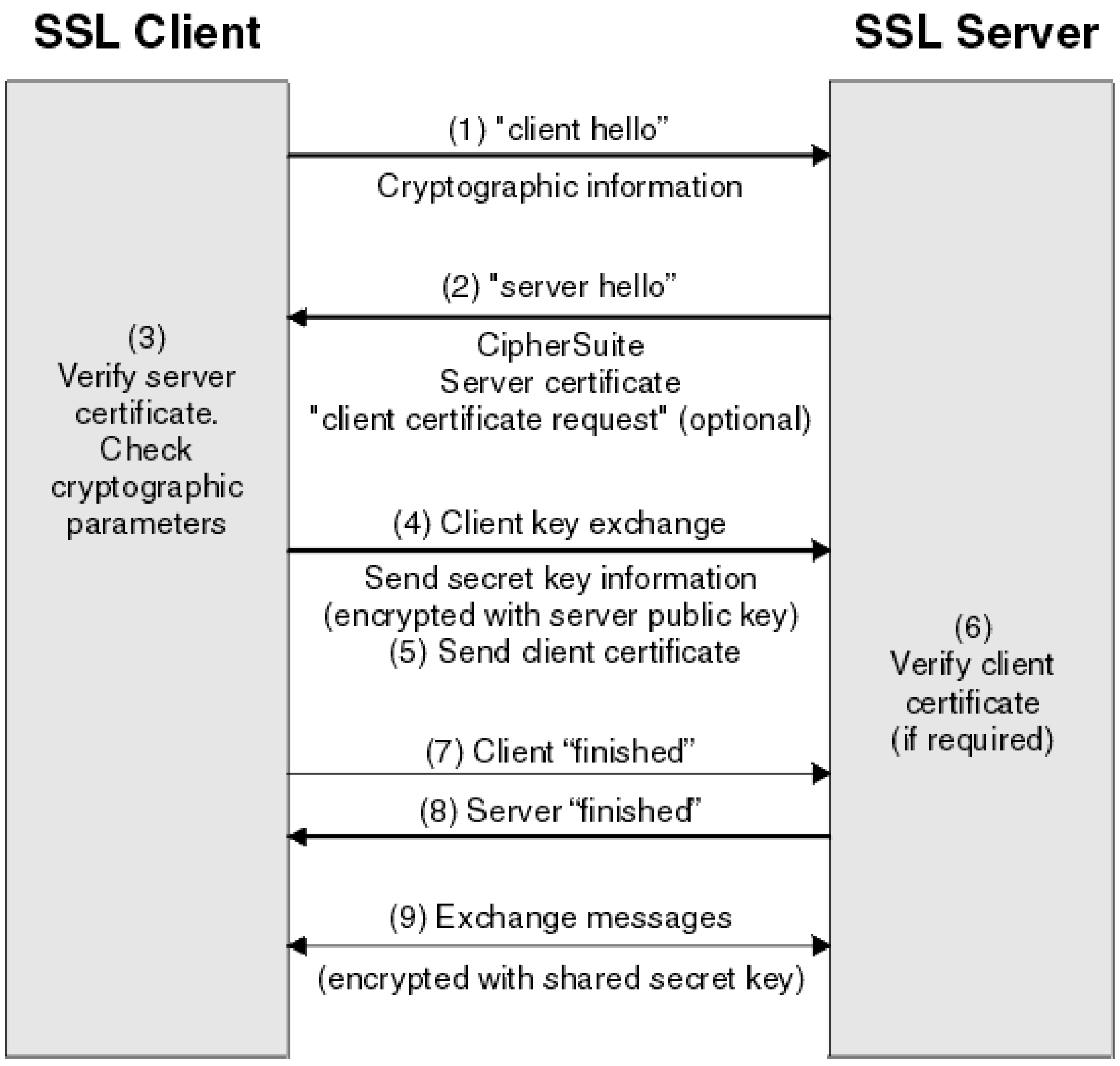
4. The SSL or TLS client sends the random byte string that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The random byte string itself is encrypted with the server's public key.

5. If the SSL or TLS server sent a client certificate request, the client sends a random byte string encrypted with the client's private key, together with the client's digital

certificate, or a no digital certificate alert. This alert is only a warning, but with some implementations the handshake fails if client authentication is mandatory.

6. The SSL or TLS server verifies the client's certificate. For more information, see How SSL and TLS provide identification, authentication, confidentiality, and integrity.
7. The SSL or TLS client sends the server a finished message, which is encrypted with the secret key, indicating that the client part of the handshake is complete.
8. The SSL or TLS server sends the client a finished message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.
9. For the duration of the SSL or TLS session, the server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

Figure 11: TLS Key Exchange Sequence [8]


SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information
(encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

ProVerif Verification

```
set traceDisplay = long.

query attacker(ClientPrivateKey).
query attacker(ServerPrivateKey).
query attacker(ServerMKey).
query attacker(ClientMKey).

query attacker(DecClientFinished).

query x:bitstring; event(startConnection(x)) ==> event(acceptConnection(x)
   ).
query x:bitstring; inj-event(startConnection(x)) ==> inj-event(
   acceptConnection(x)).

free c:channel.

type nonce.
type certificate.
type key.
type secretkey.
free DecClientFinished:bitstring[private].
free ClientCertificate:certificate[private].
free ServerCertificate:certificate[private].
free ValidCANames:certificate[private].
type mkey.
free ServerMKey:mkey [private].
free ClientMKey:mkey [private].
fun MKeyGeneration(bitstring, key, key):mkey.
event ClientMKeyGeneration(mkey).
event ServerMKeyGeneration(mkey).
fun Encryption(bitstring, mkey):nonce.
reduc  forall x:bitstring, y:mkey; Decryption(Encryption(x,y),y)=x.

free ClientHello:bitstring[private].
free ServerHello:bitstring[private].
free ClientCertificateRequest:bitstring[private].

free ClientPrivateKey:key[private].
free ClientPublicKey:key[private].
free ServerPrivateKey:key[private].
free ServerPublicKey:key[private].
free randomByteForKey:bitstring[private].
free EncryptedrandomByteForKey:nonce[private].
free DecryptedrandomByteForKey:bitstring[private].
free ClientFinished:bitstring[private].
free ServerFinished:bitstring[private].

free ComputedSecretKeyClient:secretkey[private].
free ComputedSecretKeyServer:secretkey[private].

event ClientFinishDecryption(bitstring).
event VerifyServerCertificate(certificate).
event VerifyClientCertificate(certificate).
event RandomByteDecryption(bitstring).
event ClientComputation(secretkey).
event ServerComputation(secretkey).
event startConnection(bitstring).
event acceptConnection(bitstring).

let Client =
  new ClientHello:bitstring;
  out(c, ClientHello).

let Server =
  in(c, ClientHello:bitstring);
  new ServerCertificate:certificate;
  new ServerHello:bitstring;
  new ClientCertificateRequest:bitstring;
  out(c, (ServerHello, (ServerCertificate, ClientCertificateRequest))).

let Client2 =
  in(c, (ServerHello:bitstring, (ServerCertificate:certificate,
     ClientCertificateRequest:bitstring)));
  new ClientCertificate:certificate;
  new ValidCANames:certificate;
  let ServerCertificate = ValidCANames in
    event VerifyServerCertificate(ServerCertificate);
  if ServerCertificate = ValidCANames then
  (
    new randomByteForKey:bitstring;
    let ClientMKey = MKeyGeneration(ServerHello, ServerPublicKey,
       ClientPrivateKey) in
      event ClientMKeyGeneration(ClientMKey);
    out(c, (ClientCertificate,Encryption(randomByteForKey, ClientMKey)))
    )
    else
    (
      0
      ).

  let Server2 =
    in(c, (ClientCertificate:certificate,EncryptedrandomByteForKey:nonce))
       ;
    let ClientCertificate = ValidCANames in
      event VerifyClientCertificate(ClientCertificate);
    if ClientCertificate = ValidCANames then
    (
      new ClientPublicKey:key;
      let ServerMKey = MKeyGeneration(ClientHello, ClientPublicKey,
          ServerPrivateKey) in
        event ServerMKeyGeneration(ServerMKey);
      let DecryptedrandomByteForKey = Decryption(EncryptedrandomByteForKey
          , ServerMKey) in
        event RandomByteDecryption(DecryptedrandomByteForKey);
      event startConnection(DecryptedrandomByteForKey)
      )
      else
      (
        0
        ).
    let Client3 =
      out(c, Encryption(ClientFinished, ClientMKey)).

    let Server3 =
      in(c, EncryptedClientFinished:nonce);
      let DecClientFinished = Decryption(EncryptedClientFinished,
          ServerMKey) in
        event ClientFinishDecryption(DecClientFinished);
      if DecClientFinished = ClientFinished then
      (
        event acceptConnection(DecClientFinished)
        )
        else
        (
          0
          ).
process
  (
    !Client|!Server|!Client2|!Server2|!Client3|!Server3
    )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}new ClientHello_16: bitstring;
    {3}out(c, ClientHello_16)
) | (
    {4}!
    {5}in(c, ClientHello_17: bitstring);
    {6}new ServerCertificate_18: certificate;
    {7}new ServerHello_19: bitstring;
    {8}new ClientCertificateRequest_20: bitstring;
    {9}out(c, (ServerHello_19,(ServerCertificate_18,
       ClientCertificateRequest_20)))
) | (
    {10}!
    {11}in(c, (ServerHello_21: bitstring,(ServerCertificate_22:
       certificate,ClientCertificateRequest_23: bitstring)));
    {12}new ClientCertificate_24: certificate;
    {13}new ValidCANames_25: certificate;
    {14}let ServerCertificate_26: certificate = ValidCANames_25 in
    {15}event VerifyServerCertificate(ServerCertificate_26);
    {16}if (ServerCertificate_26 = ValidCANames_25) then
    {17}new randomByteForKey_27: bitstring;
    {18}let ClientMKey_28: mkey = MKeyGeneration(ServerHello_21,
       ServerPublicKey,ClientPrivateKey) in
    {19}event ClientMKeyGeneration(ClientMKey_28);
    {20}out(c, (ClientCertificate_24,Encryption(randomByteForKey_27,
       ClientMKey_28)))
) | (
    {21}!
    {22}in(c, (ClientCertificate_29: certificate,
       EncryptedrandomByteForKey_30: nonce));
    {23}let ClientCertificate_31: certificate = ValidCANames in
    {24}event VerifyClientCertificate(ClientCertificate_31);
    {25}if (ClientCertificate_31 = ValidCANames) then
    {26}new ClientPublicKey_32: key;
    {27}let ServerMKey_33: mkey = MKeyGeneration(ClientHello,
       ClientPublicKey_32,ServerPrivateKey) in
    {28}event ServerMKeyGeneration(ServerMKey_33);
    {29}let DecryptedrandomByteForKey_34: bitstring = Decryption(
        EncryptedrandomByteForKey_30,ServerMKey_33) in
    {30}event RandomByteDecryption(DecryptedrandomByteForKey_34);
    {31}event startConnection(DecryptedrandomByteForKey_34)
) | (
    {32}!
    {33}out(c, Encryption(ClientFinished,ClientMKey))
) | (
    {34}!
    {35}in(c, EncryptedClientFinished: nonce);
    {36}let DecClientFinished_35: bitstring = Decryption(
        EncryptedClientFinished,ServerMKey) in
    {37}event ClientFinishDecryption(DecClientFinished_35);
    {38}if (DecClientFinished_35 = ClientFinished) then
    {39}event acceptConnection(DecClientFinished_35)
)

-- Query not attacker(ClientPrivateKey[])
Completing...
Starting query not attacker(ClientPrivateKey[])
RESULT not attacker(ClientPrivateKey[]) is true.
-- Query not attacker(ServerPrivateKey[])
Completing...
Starting query not attacker(ServerPrivateKey[])
RESULT not attacker(ServerPrivateKey[]) is true.
-- Query not attacker(ServerMKey[])
Completing...
Starting query not attacker(ServerMKey[])
RESULT not attacker(ServerMKey[]) is true.
-- Query not attacker(ClientMKey[])
Completing...
Starting query not attacker(ClientMKey[])
RESULT not attacker(ClientMKey[]) is true.
-- Query not attacker(DecClientFinished[])
Completing...
Starting query not attacker(DecClientFinished[])
RESULT not attacker(DecClientFinished[]) is true.
-- Query event(startConnection(x_36)) ==> event(acceptConnection(x_36))
Completing...
Starting query event(startConnection(x_36)) ==> event(acceptConnection(
    x_36))
RESULT event(startConnection(x_36)) ==> event(acceptConnection(x_36)) is
    true.
```

```
-- Query inj-event(startConnection(x_37)) ==> inj-event(acceptConnection(
    x_37))
Completing...
Starting query inj-event(startConnection(x_37)) ==> inj-event(
    acceptConnection(x_37))
RESULT inj-event(startConnection(x_37)) ==> inj-event(acceptConnection(
    x_37)) is true.
```

Appendix 9 – PEAP-MSCHAPv2 Authentication

MS-CHAP is the Microsoft version of the Challenge-Handshake Authentication Protocol, CHAP. The protocol exists in two versions, MS-CHAPv1 and MS-CHAPv2. PEAP-MSCHAP provides an encrypted and authenticated tunnel based on TLS. In the first phase, a TLS session is negotiated and established. The client also authenticates the server by using a certificate. Optionally, the server can also authenticate the client. In the second phase, EAP messages are encrypted by using the key negotiated in phase one. The basic idea of PEAP and EAP-TTLS are identical. However, PEAP can only use EAP protocols in the second phase, while EAP-TTLS can use EAP or non-EAP protocols. [147]
MS-CHAP is used as one authentication option in Microsoft's implementation of the PPTP protocol for virtual private networks. It is also used as an authentication option with RADIUS[2] servers which are used with IEEE 802.1X (e.g., Wi-Fi security using the WPA-Enterprise protocol). It is further used as the main authentication option of the Protected Extensible Authentication Protocol (PEAP). [147] Compared with CHAP MS-CHAP is enabled by negotiating CHAP Algorithm, Authentication Protocol provides an authenticator-controlled password change mechanism, which provides an authenticator-controlled authentication retry mechanism that failure codes returned in the failure packet message field. MS-CHAPv2 provides mutual authentication between peers with a peer challenge on the response packet and an authenticator response on the success packet. The Extensible Authentication Protocol method for Microsoft Challenge Handshake Authentication Protocol (CHAP) is an EAP method that is designed to meet this need. It does so by having the client and server use MSCHAPv2 to mutually authenticate each other. The flow for successful authentication with Extensible Authentication Protocol Method for Microsoft CHAP is as follows: [147]

1. An EAP session is established between a client (EAP peer) and an EAP server.
2. The EAP server and EAP peer negotiate the EAP method to use. The Extensible Authentication Protocol Method for Microsoft CHAP is selected.
3. The EAP peer and EAP server continue to exchange EAP messages with MSCHAPv2 packets encapsulated in the payload.

After the MSCHAPv2 packets successfully authenticate the client and the server to each other, the EAP authentication finishes. Diffie-Hellman authentication has been described in Appendix 3 and TLS Appendix 8.

ProVerif Verification

```
set traceDisplay = long.
query attacker(ServerMKey).
query attacker(ClientMKey).
query attacker(DecryptedTraffic1).
query attacker(DecryptedTraffic2).
query x:bitstring; event(TrafficDecryption1(x)).
query x:bitstring; event(TrafficDecryption2(x)).
query x:bitstring; event(TrafficDecryption1(x)) ==> event(
    TrafficDecryption2(x)).
query x:bitstring; inj-event(TrafficDecryption1(x)) ==> inj-event(
    TrafficDecryption2(x)).

type nonce.
type key.
type certificate.
type material.
type challenge.
type response.
free DecryptedMessage:bitstring[private].
free DecryptedMessage2:bitstring[private].
free c:channel [private].
free ValidCA:certificate[private].
free EncKey:material[private].
free MSCHAPv2Challenge:challenge[private].
free Success:nonce [private].
free EncryptionKey, ClientKey:key[private].
free EAPResponse, EAPAuthRequest:bitstring[private].
free ClientDone, ClientKeyExchange, Hello, HelloAck, DecryptedTraffic1,
    DecryptedTraffic2:bitstring[private].
free ClientIdentity, DBClientIdentity, IdentityExists,
    AuthenticationComplete, AuthenticationCompleteACK:bitstring [private].
free ServerKeyExchange, ServerDone, ServerHello, Cipher, ServerFinished:
    bitstring [private].
free ServerCertificate, ClientCertificate:certificate[private].
free MSCHAPPassword:bitstring [private].
fun SolveChallenge(challenge, bitstring):response.

type mkey.
free ServerMKey:mkey [private].
free ClientMKey:mkey [private].
```

```
fun MKeyGeneration(bitstring, key, key):mkey.
event ClientMKeyGeneration(mkey).
event ServerMKeyGeneration(mkey).

fun GenerateKey(material):key.
fun EncryptTraffic(bitstring, mkey):nonce.
reduc forall x:bitstring, y:mkey; DecryptTraffic(EncryptTraffic(x,y),y)=x.
fun CombinedAuthentication(bitstring, response):nonce.

free ServerPublicKey, ServerPrivateKey, ClientPublicKey, ClientPrivateKey
    : key [private].
free TLSHello:bitstring [private].
event DBClientIdentityCheck(bitstring).
event FirstChallenge(response).
event EncryptionKeyGeneration(key).
event Decrypt1(bitstring).
event Decrypt2(bitstring).
event Authentication(nonce).
event DCAuthentication(nonce).
event ServerCertificateValidation(certificate).
event ClientCertificateValidation(certificate).
event TrafficDecryption1(bitstring).
event TrafficDecryption2(bitstring).
event ClientPrivateKeyGeneration(key).

let AP1 =
  new AuthenticateRequest:bitstring;
  out(c, AuthenticateRequest).

let Client1 =
  in(c, AuthenticateRequest:bitstring);
  new ClientIdentity:bitstring;
  out(c, ClientIdentity).

let AP2 =
  in(c, ClientIdentity:bitstring);
  out(c, ClientIdentity).

let Server1 =
  in(c, ClientIdentity:bitstring);
  let ClientIdentity = DBClientIdentity in
    event DBClientIdentityCheck(ClientIdentity);
  if ClientIdentity = DBClientIdentity then
  (
    out(c, IdentityExists)
    )
    else
    (
      0
      ).
let AP3 =
  in(c, IdentityExists:bitstring);
  new EAPAuthRequest:bitstring;
  out(c, EAPAuthRequest).

let Client2 =
  in(c, EAPAuthRequest:bitstring);
  new TLSHello:bitstring;
  out(c, TLSHello).

let AP4 =
  in(c, TLSHello:bitstring);
  out(c, TLSHello).

let Server2 =
  in(c, TLSHello:bitstring);
  out(c, (ServerHello, (ServerCertificate, (ServerKeyExchange, ServerDone)
     ))).

let AP5 =
  in(c, (ServerHello:bitstring, (ServerCertificate:certificate, (
     ServerKeyExchange:bitstring, ServerDone:bitstring))));
  out(c, (ServerHello, (ServerCertificate, (ServerKeyExchange, ServerDone)
     ))).

let Client3 =
  in(c, (ServerHello:bitstring, (ServerCertificate:certificate, (
     ServerKeyExchange:bitstring, ServerDone:bitstring))));
  let ServerCertificate = ValidCA in
    event ServerCertificateValidation(ServerCertificate);
  if ServerCertificate = ValidCA then
  (
    out(c, (ClientKeyExchange, (Cipher, (ClientCertificate, ClientDone))))
    )
    else
    (
      0
      ).

let AP6 =
  in(c, (ClientKeyExchange:bitstring, (Cipher:bitstring, (
     ClientCertificate:certificate, ClientFinished:bitstring))));
  out(c, (ClientKeyExchange, (Cipher, (ClientCertificate, ClientDone)))).

let Server3 =
  in(c, (ClientKeyExchange:bitstring, (Cipher:bitstring, (
     ClientCertificate:certificate, ClientFinished:bitstring))));
  let ClientCertificate = ValidCA in
    event ClientCertificateValidation(ClientCertificate);
  if ClientCertificate = ValidCA then
  (
    out(c, (Cipher, ServerFinished))
    )
    else
    (
      0
      ).

let AP7 =
  in(c, (Cipher:bitstring, ServerFinished:bitstring));
  out(c, (Cipher, ServerFinished)).

let Client4 =
  in(c, (Cipher:bitstring, ServerFinished:bitstring));
  out(c, EAPResponse).

let AP8 =
  in(c, EAPResponse:bitstring);
  out(c, EAPResponse).

let Server4 =
  in(c, EAPResponse:bitstring);
  out(c, EAPAuthRequest).

let AP9 =
  in(c, EAPAuthRequest:bitstring);
  out(c, EAPAuthRequest).

let Client5 =
  in(c, EAPAuthRequest:bitstring);
  out(c, ClientIdentity).

let AP10 =
  in(c, ClientIdentity:bitstring);
  out(c, ClientIdentity).

let Server5 =
  in(c, ClientIdentity:bitstring);
  let ClientIdentity = DBClientIdentity in
    event DBClientIdentityCheck(ClientIdentity);
  if ClientIdentity = DBClientIdentity then
  (
    out(c, MSCHAPv2Challenge)
    )
    else
    (
      0
      ).
let AP11 =
  in(c, MSCHAPv2Challenge:challenge);
  out(c, MSCHAPv2Challenge).

let Client6 =
  in(c, MSCHAPv2Challenge:challenge);
  let SolvedChallenge = SolveChallenge(MSCHAPv2Challenge, MSCHAPPassword)
      in
    event FirstChallenge(SolvedChallenge);
  out(c, SolvedChallenge).

let AP12 =
  in(c, SolvedChallenge:response);
  out(c, SolvedChallenge).

let Server6 =
  in(c, SolvedChallenge:response);
  if SolvedChallenge = SolveChallenge(MSCHAPv2Challenge, MSCHAPPassword)
      then
  (
    let Success = CombinedAuthentication(ClientIdentity, SolvedChallenge)
        in
      event Authentication(Success);
    let DBSuccess = Success in
      event DCAuthentication(DBSuccess);
    if Success = DBSuccess then
    (
      out(c, AuthenticationComplete)
      )
    else
    (
      0
      )).

let AP13 =
  in(c, AuthenticationComplete:bitstring);
  out(c, AuthenticationComplete).

let Client7 =
  in(c, AuthenticationComplete:bitstring);
  out(c, AuthenticationCompleteACK).

let AP14 =
  in(c, AuthenticationCompleteACK:bitstring);
  out(c, AuthenticationCompleteACK).

let Server7 =
  in(c, AuthenticationCompleteACK:bitstring);
  out(c, EncKey).

let AP15 =
  in(c, EncKey:material);
  out(c, EncKey).

let Client8 =
  in(c, EncKey:material);
  let ClientPrivateKey = GenerateKey(EncKey) in
    event ClientPrivateKeyGeneration(ClientPrivateKey);
  let ClientMKey = MKeyGeneration(ServerHello, ServerPublicKey,
      ClientPrivateKey) in
    event ClientMKeyGeneration(ClientMKey);
  out(c, EncryptTraffic(Hello, ClientMKey)).
```

```
let AP16 =
  in(c, EncryptedTraffic1:nonce);
  out(c, EncryptedTraffic1).

let Server8 =
  in(c, EncryptedTraffic1:nonce);
  let ServerMKey = MKeyGeneration(TLSHello, ClientPublicKey,
     ServerPrivateKey) in
    event ServerMKeyGeneration(ServerMKey);
  let DecryptedTraffic1 = DecryptTraffic(EncryptedTraffic1, ServerMKey) in
    event TrafficDecryption1(DecryptedTraffic1);
  out(c, EncryptTraffic(HelloAck, ServerMKey)).

let AP17 =
  in(c, EncryptedTraffic2:nonce);
  out(c, EncryptedTraffic2).

let Client9 =
  in(c, EncryptedTraffic2:nonce);
  let DecryptedTraffic2 = DecryptTraffic(EncryptedTraffic2, ClientMKey) in
    event TrafficDecryption2(DecryptedTraffic2).

  process
  (
    !AP1|!Client1|!AP2|!Server1|!AP3|!Client2|!AP4|!Server2|!AP5|!Client3
       |!AP6|!Server3|!AP7|!Client4|!AP8|!Server4|!AP9|!Client5|!AP10|!
       Server5|!AP11|!Client6|!AP12|!Server6|!AP13|!Client7|!AP14|!Server7
       |!AP15|!Client8|
    !AP16|!Server8|!AP17|!Client9
    )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}new AuthenticateRequest: bitstring;
    {3}out(c, AuthenticateRequest)
) | (
    {4}!
    {5}in(c, AuthenticateRequest_16: bitstring);
    {6}new ClientIdentity_17: bitstring;
    {7}out(c, ClientIdentity_17)
) | (
    {8}!
    {9}in(c, ClientIdentity_18: bitstring);
    {10}out(c, ClientIdentity_18)
) | (
    {11}!
    {12}in(c, ClientIdentity_19: bitstring);
    {13}let ClientIdentity_20: bitstring = DBClientIdentity in
    {14}event DBClientIdentityCheck(ClientIdentity_20);
    {15}if (ClientIdentity_20 = DBClientIdentity) then
    {16}out(c, IdentityExists)
) | (
    {17}!
    {18}in(c, IdentityExists_21: bitstring);
    {19}new EAPAuthRequest_22: bitstring;
    {20}out(c, EAPAuthRequest_22)
) | (
    {21}!
    {22}in(c, EAPAuthRequest_23: bitstring);
    {23}new TLSHello_24: bitstring;
    {24}out(c, TLSHello_24)
) | (
    {25}!
    {26}in(c, TLSHello_25: bitstring);
    {27}out(c, TLSHello_25)
) | (
    {28}!
    {29}in(c, TLSHello_26: bitstring);
    {30}out(c, (ServerHello,(ServerCertificate,(ServerKeyExchange,
       ServerDone))))
```

```
) | (
    {31}!
    {32}in(c, (ServerHello_27: bitstring,(ServerCertificate_28:
        certificate,(ServerKeyExchange_29: bitstring,ServerDone_30:
        bitstring))));
    {33}out(c, (ServerHello_27,(ServerCertificate_28,(ServerKeyExchange_29
        ,ServerDone_30))))
) | (
    {34}!
    {35}in(c, (ServerHello_31: bitstring,(ServerCertificate_32:
        certificate,(ServerKeyExchange_33: bitstring,ServerDone_34:
        bitstring))));
    {36}let ServerCertificate_35: certificate = ValidCA in
    {37}event ServerCertificateValidation(ServerCertificate_35);
    {38}if (ServerCertificate_35 = ValidCA) then
    {39}out(c, (ClientKeyExchange,(Cipher,(ClientCertificate,ClientDone)))
        )
) | (
    {40}!
    {41}in(c, (ClientKeyExchange_36: bitstring,(Cipher_37: bitstring,(
        ClientCertificate_38: certificate,ClientFinished: bitstring))));
    {42}out(c, (ClientKeyExchange_36,(Cipher_37,(ClientCertificate_38,
        ClientDone))))
) | (
    {43}!
    {44}in(c, (ClientKeyExchange_39: bitstring,(Cipher_40: bitstring,(
        ClientCertificate_41: certificate,ClientFinished_42: bitstring))));
    {45}let ClientCertificate_43: certificate = ValidCA in
    {46}event ClientCertificateValidation(ClientCertificate_43);
    {47}if (ClientCertificate_43 = ValidCA) then
    {48}out(c, (Cipher_40,ServerFinished))
) | (
    {49}!
    {50}in(c, (Cipher_44: bitstring,ServerFinished_45: bitstring));
    {51}out(c, (Cipher_44,ServerFinished_45))
) | (
    {52}!
    {53}in(c, (Cipher_46: bitstring,ServerFinished_47: bitstring));
    {54}out(c, EAPResponse)
) | (
    {55}!
    {56}in(c, EAPResponse_48: bitstring);
    {57}out(c, EAPResponse_48)
) | (
    {58}!
    {59}in(c, EAPResponse_49: bitstring);
    {60}out(c, EAPAuthRequest)
) | (
    {61}!
    {62}in(c, EAPAuthRequest_50: bitstring);
    {63}out(c, EAPAuthRequest_50)
) | (
    {64}!
    {65}in(c, EAPAuthRequest_51: bitstring);
    {66}out(c, ClientIdentity)
) | (
    {67}!
    {68}in(c, ClientIdentity_52: bitstring);
    {69}out(c, ClientIdentity_52)
) | (
    {70}!
    {71}in(c, ClientIdentity_53: bitstring);
    {72}let ClientIdentity_54: bitstring = DBClientIdentity in
    {73}event DBClientIdentityCheck(ClientIdentity_54);
    {74}if (ClientIdentity_54 = DBClientIdentity) then
    {75}out(c, MSCHAPv2Challenge)
) | (
    {76}!
    {77}in(c, MSCHAPv2Challenge_55: challenge);
    {78}out(c, MSCHAPv2Challenge_55)
) | (
    {79}!
    {80}in(c, MSCHAPv2Challenge_56: challenge);
    {81}let SolvedChallenge: response = SolveChallenge(
       MSCHAPv2Challenge_56,MSCHAPPassword) in
    {82}event FirstChallenge(SolvedChallenge);
    {83}out(c, SolvedChallenge)
) | (
    {84}!
    {85}in(c, SolvedChallenge_57: response);
    {86}out(c, SolvedChallenge_57)
) | (
    {87}!
    {88}in(c, SolvedChallenge_58: response);
    {89}if (SolvedChallenge_58 = SolveChallenge(MSCHAPv2Challenge,
       MSCHAPPassword)) then
    {90}let Success_59: nonce = CombinedAuthentication(ClientIdentity,
       SolvedChallenge_58) in
    {91}event Authentication(Success_59);
    {92}let DBSuccess: nonce = Success_59 in
    {93}event DCAuthentication(DBSuccess);
    {94}if (Success_59 = DBSuccess) then
    {95}out(c, AuthenticationComplete)
) | (
    {96}!
    {97}in(c, AuthenticationComplete_60: bitstring);
    {98}out(c, AuthenticationComplete_60)
) | (
    {99}!
    {100}in(c, AuthenticationComplete_61: bitstring);
    {101}out(c, AuthenticationCompleteACK)
) | (
    {102}!
    {103}in(c, AuthenticationCompleteACK_62: bitstring);
    {104}out(c, AuthenticationCompleteACK_62)
) | (
    {105}!
    {106}in(c, AuthenticationCompleteACK_63: bitstring);
    {107}out(c, EncKey)
) | (
    {108}!
    {109}in(c, EncKey_64: material);
    {110}out(c, EncKey_64)
) | (
    {111}!
    {112}in(c, EncKey_65: material);
    {113}let ClientPrivateKey_66: key = GenerateKey(EncKey_65) in
    {114}event ClientPrivateKeyGeneration(ClientPrivateKey_66);
    {115}let ClientMKey_67: mkey = MKeyGeneration(ServerHello,
       ServerPublicKey,ClientPrivateKey_66) in
    {116}event ClientMKeyGeneration(ClientMKey_67);
    {117}out(c, EncryptTraffic(Hello,ClientMKey_67))
) | (
    {118}!
    {119}in(c, EncryptedTraffic1: nonce);
    {120}out(c, EncryptedTraffic1)
```

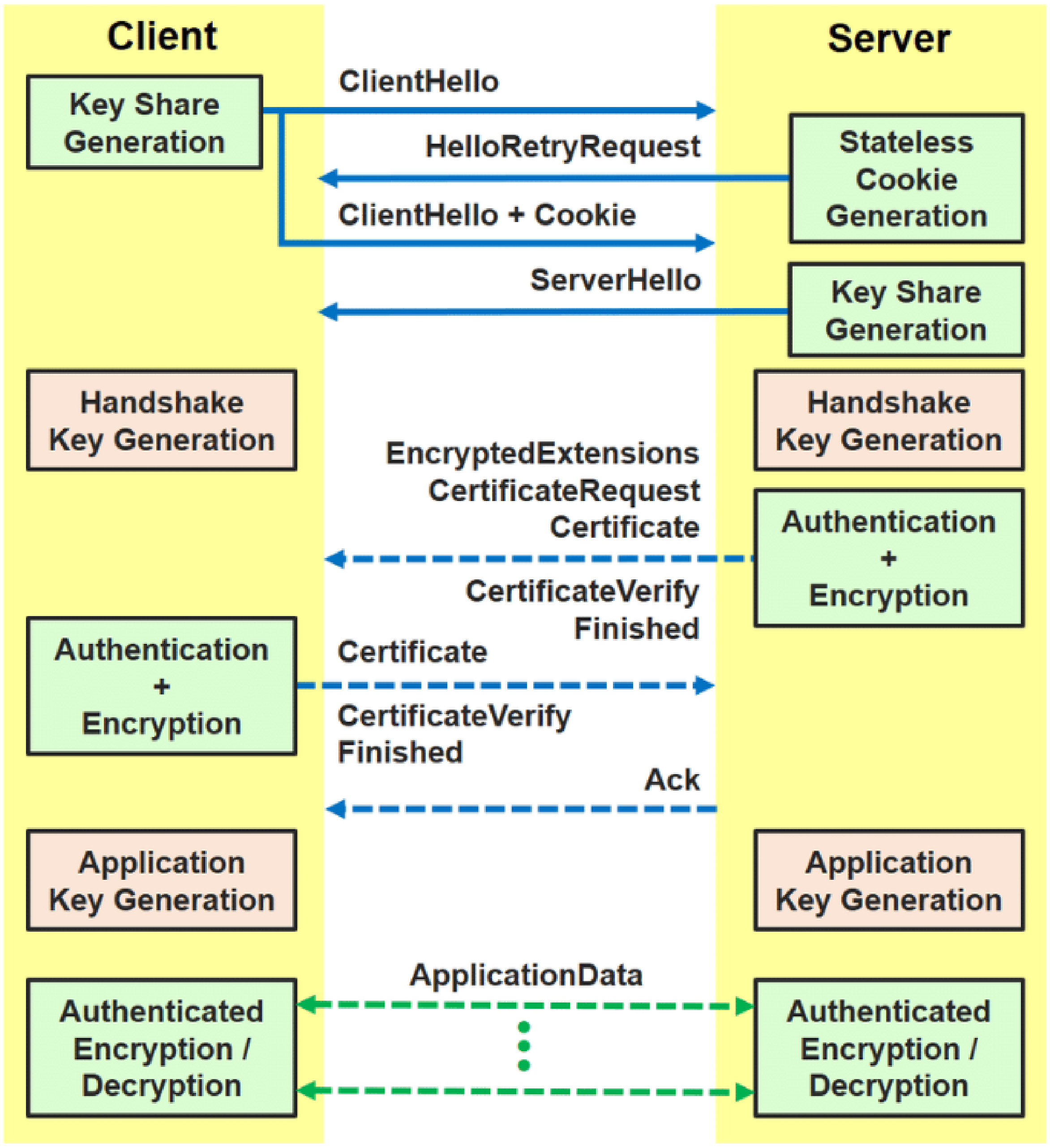
```
) | (
    {121}!
    {122}in(c, EncryptedTraffic1_68: nonce);
    {123}let ServerMKey_69: mkey = MKeyGeneration(TLSHello,ClientPublicKey
        ,ServerPrivateKey) in
    {124}event ServerMKeyGeneration(ServerMKey_69);
    {125}let DecryptedTraffic1_70: bitstring = DecryptTraffic(
        EncryptedTraffic1_68,ServerMKey_69) in
    {126}event TrafficDecryption1(DecryptedTraffic1_70);
    {127}out(c, EncryptTraffic(HelloAck,ServerMKey_69))
) | (
    {128}!
    {129}in(c, EncryptedTraffic2: nonce);
    {130}out(c, EncryptedTraffic2)
) | (
    {131}!
    {132}in(c, EncryptedTraffic2_71: nonce);
    {133}let DecryptedTraffic2_72: bitstring = DecryptTraffic(
        EncryptedTraffic2_71,ClientMKey) in
    {134}event TrafficDecryption2(DecryptedTraffic2_72)
)

-- Query not attacker(ServerMKey[])
nounif mess(c[],AuthenticateRequest_144)/-5000
Completing...
Starting query not attacker(ServerMKey[])
RESULT not attacker(ServerMKey[]) is true.
-- Query not attacker(ClientMKey[])
nounif mess(c[],AuthenticateRequest_948)/-5000
Completing...
Starting query not attacker(ClientMKey[])
RESULT not attacker(ClientMKey[]) is true.
-- Query not attacker(DecryptedTraffic1[])
nounif mess(c[],AuthenticateRequest_1748)/-5000
Completing...
Starting query not attacker(DecryptedTraffic1[])
RESULT not attacker(DecryptedTraffic1[]) is true.
-- Query not attacker(DecryptedTraffic2[])
nounif mess(c[],AuthenticateRequest_2548)/-5000
Completing...
Starting query not attacker(DecryptedTraffic2[])
RESULT not attacker(DecryptedTraffic2[]) is true.
```

```
-- Query not event(TrafficDecryption1(x_73))
nounif mess(c[],AuthenticateRequest_3348)/-5000
Completing...
Starting query not event(TrafficDecryption1(x_73))
RESULT not event(TrafficDecryption1(x_73)) is true.
-- Query not event(TrafficDecryption2(x_74))
nounif mess(c[],AuthenticateRequest_4155)/-5000
Completing...
Starting query not event(TrafficDecryption2(x_74))
RESULT not event(TrafficDecryption2(x_74)) is true.
-- Query event(TrafficDecryption1(x_75)) ==> event(TrafficDecryption2(x_75
   ))
nounif mess(c[],AuthenticateRequest_4961)/-5000
Completing...
Starting query event(TrafficDecryption1(x_75)) ==> event(
   TrafficDecryption2(x_75))
RESULT event(TrafficDecryption1(x_75)) ==> event(TrafficDecryption2(x_75))
    is true.
-- Query inj-event(TrafficDecryption1(x_76)) ==> inj-event(
   TrafficDecryption2(x_76))
nounif mess(c[],AuthenticateRequest_5768)/-5000
Completing...
Starting query inj-event(TrafficDecryption1(x_76)) ==> inj-event(
   TrafficDecryption2(x_76))
RESULT inj-event(TrafficDecryption1(x_76)) ==> inj-event(
   TrafficDecryption2(x_76)) is true.
```

Appendix 10 – DTLS Authentication

DTLS uses TLS based key exchange, which is described in Appendix 8. Difference between DTLS and TLS is that which transport protocol they use, TLS uses TCP and DTLS UDP. Message flow for this authentication on key exchange in Figure 12 is as follows:

Figure 12: DTLS Key Exchange Sequence [9]



ProVerif Verification

```
set traceDisplay = long.

query attacker(ServerPrivKey).
query attacker(ClientPrivKey).
query attacker(ServerMKey).
query attacker(ClientMKey).
query attacker(DecryptedData).
query attacker(DecryptedData2).

query x:bitstring; event(DataDecryption1(x)).
query x:bitstring; event(DataDecryption2(x)).
query x:bitstring; event(DataDecryption1(x)) ==> event(DataDecryption2(x))
   .
query x:bitstring; inj-event(DataDecryption1(x)) ==> inj-event(
   DataDecryption2(x)).

type certificate.
type key.
type nonce.
type mkey.

free c: channel.
free ClientCertificate, ServerCertificate, ValidCA:certificate[private].
free Hello, TestHello, TestHelloAck, HelloVerify, Hello2, DecryptedData2,
   DecryptedData:bitstring [private].
free ServerHello, ServerFinished, ServerKeyExchange, ClientKeyExchange,
   ClientCertificateRequest, Cipher, ClientFinished, ServerHelloDone :
   bitstring[private].
free ServerPubKey,ServerPrivKey, ClientPrivKey, ClientPubKey : key [
   private].

fun PublicKeyGeneration(certificate):key.
fun Encryption(bitstring, mkey):nonce.
reduc  forall x:bitstring, y:mkey; Decryption(Encryption(x,y),y)=x.


free ServerMKey:mkey [private].
free ClientMKey:mkey [private].
fun CommunicationKeyGeneration(bitstring, key, key):mkey.
event ClientMKeyGeneration(mkey).
event ServerMKeyGeneration(mkey).

event HelloVerification(bitstring).
event ClientCertificateValidation(certificate).
event ServerCertificateValidation(certificate).
event ClientPublicKeyGeneration(key).
event ServerPublicKeyGeneration(key).
event DataDecryption1(bitstring).
event DataDecryption2(bitstring).
event DataDecryption3(bitstring).

let Client1 =
  out(c, Hello).

let Server1 =
  in(c, Hello:bitstring);
  out(c, HelloVerify).

let Client2 =
  in(c, HelloVerify:bitstring);
  out(c, Hello2).

let Server2 =
  in(c, Hello2:bitstring);
  let Hello2 = Hello in
    event HelloVerification(Hello2);
  if Hello = Hello2 then
  (
    out(c, (ServerHello, (ServerCertificate, (ServerKeyExchange, (
        ClientCertificateRequest, ServerHelloDone))))
    )
    else
    (
      0
      ).

let Client3 =
  in(c, (ServerHello:bitstring, (ServerCertificate:certificate, (
      ServerKeyExchange:bitstring, (ClientCertificateRequest:bitstring,
      ServerHelloDone:bitstring)))));
  let ServerCertificate = ValidCA in
    event ServerCertificateValidation(ServerCertificate);
  if ServerCertificate = ValidCA then
  (
    out(c, (ClientCertificate, (ClientKeyExchange, (Cipher, ClientFinished
       ))))
    )
    else
    (
      0
      ).
let Server3 =
  in(c, (ClientCertificate:certificate, (ClientKeyExchange:bitstring, (
     Cipher:bitstring, ClientFinished:bitstring))));
  let ClientCertificate = ValidCA in
    event ClientCertificateValidation(ClientCertificate);
  if ClientCertificate = ValidCA then
  (
    out(c, (Cipher, ServerFinished))
    )
    else
    (
      0
      ).
let Client4 =
  in(c, (Cipher:bitstring, ServerFinished:bitstring));
  let ServerPubKey = PublicKeyGeneration(ServerCertificate) in
    event ServerPublicKeyGeneration(ServerPubKey);
  let ClientMKey = CommunicationKeyGeneration(HelloVerify, ServerPubKey,
     ClientPrivKey) in
    event ClientMKeyGeneration(ClientMKey);
  out(c, Encryption(TestHello, ClientMKey)).

let Server4 =
  in(c, EncryptedData:nonce);
  let ClientPubKey = PublicKeyGeneration(ClientCertificate) in
    event ClientPublicKeyGeneration(ClientPubKey);
  let ServerMKey = CommunicationKeyGeneration(Hello, ClientPubKey,
     ServerPrivKey) in
    event ClientMKeyGeneration(ServerMKey);
  let DecryptedData = Decryption(EncryptedData, ServerMKey) in
    event DataDecryption1(DecryptedData);
  if DecryptedData = TestHello then
  (
    out(c, Encryption(TestHelloAck, ServerMKey))
    )
    else
    (
      0
      ).

let Client5 =
  in(c, EncryptedData2:nonce);
  let DecryptedData2 = Decryption(EncryptedData2, ClientMKey) in
    event DataDecryption2(DecryptedData2).

process(
  !Client1|!Server1|!Client2|!Server2|!Client3|!Server3|!Client4|!Server4
     |!Client5
  )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, Hello)
) | (
    {3}!
    {4}in(c, Hello_16: bitstring);
    {5}out(c, HelloVerify)
) | (
    {6}!
    {7}in(c, HelloVerify_17: bitstring);
    {8}out(c, Hello2)
) | (
    {9}!
    {10}in(c, Hello2_18: bitstring);
    {11}let Hello2_19: bitstring = Hello in
    {12}event HelloVerification(Hello2_19);
    {13}if (Hello = Hello2_19) then
    {14}out(c, (ServerHello,(ServerCertificate,(ServerKeyExchange,(
       ClientCertificateRequest,ServerHelloDone)))))
) | (
    {15}!
    {16}in(c, (ServerHello_20: bitstring,(ServerCertificate_21:
       certificate,(ServerKeyExchange_22: bitstring,(
       ClientCertificateRequest_23: bitstring,ServerHelloDone_24:
       bitstring)))));
    {17}let ServerCertificate_25: certificate = ValidCA in
    {18}event ServerCertificateValidation(ServerCertificate_25);
    {19}if (ServerCertificate_25 = ValidCA) then
    {20}out(c, (ClientCertificate,(ClientKeyExchange,(Cipher,
       ClientFinished))))
) | (
    {21}!
    {22}in(c, (ClientCertificate_26: certificate,(ClientKeyExchange_27:
       bitstring,(Cipher_28: bitstring,ClientFinished_29: bitstring))));
    {23}let ClientCertificate_30: certificate = ValidCA in
    {24}event ClientCertificateValidation(ClientCertificate_30);
    {25}if (ClientCertificate_30 = ValidCA) then
    {26}out(c, (Cipher_28,ServerFinished))
) | (
    {27}!
    {28}in(c, (Cipher_31: bitstring,ServerFinished_32: bitstring));
    {29}let ServerPubKey_33: key = PublicKeyGeneration(ServerCertificate)
       in
    {30}event ServerPublicKeyGeneration(ServerPubKey_33);
    {31}let ClientMKey_34: mkey = CommunicationKeyGeneration(HelloVerify,
       ServerPubKey_33,ClientPrivKey) in
    {32}event ClientMKeyGeneration(ClientMKey_34);
    {33}out(c, Encryption(TestHello,ClientMKey_34))
) | (
    {34}!
    {35}in(c, EncryptedData: nonce);
    {36}let ClientPubKey_35: key = PublicKeyGeneration(ClientCertificate)
       in
    {37}event ClientPublicKeyGeneration(ClientPubKey_35);
    {38}let ServerMKey_36: mkey = CommunicationKeyGeneration(Hello,
       ClientPubKey_35,ServerPrivKey) in
    {39}event ClientMKeyGeneration(ServerMKey_36);
    {40}let DecryptedData_37: bitstring = Decryption(EncryptedData,
       ServerMKey_36) in
    {41}event DataDecryption1(DecryptedData_37);
    {42}if (DecryptedData_37 = TestHello) then
    {43}out(c, Encryption(TestHelloAck,ServerMKey_36))
) | (
    {44}!
    {45}in(c, EncryptedData2: nonce);
    {46}let DecryptedData2_38: bitstring = Decryption(EncryptedData2,
       ClientMKey) in
    {47}event DataDecryption2(DecryptedData2_38)
)

-- Query not attacker(ServerPrivKey[])
Completing...
Starting query not attacker(ServerPrivKey[])
RESULT not attacker(ServerPrivKey[]) is true.
-- Query not attacker(ClientPrivKey[])
Completing...
Starting query not attacker(ClientPrivKey[])
RESULT not attacker(ClientPrivKey[]) is true.
-- Query not attacker(ServerMKey[])
Completing...
Starting query not attacker(ServerMKey[])
```

```
RESULT not attacker(ServerMKey[]) is true.
-- Query not attacker(ClientMKey[])
Completing...
Starting query not attacker(ClientMKey[])
RESULT not attacker(ClientMKey[]) is true.
-- Query not attacker(DecryptedData[])
Completing...
Starting query not attacker(DecryptedData[])
RESULT not attacker(DecryptedData[]) is true.
-- Query not attacker(DecryptedData2[])
Completing...
Starting query not attacker(DecryptedData2[])
RESULT not attacker(DecryptedData2[]) is true.
-- Query not event(DataDecryption1(x_39))
Completing...
Starting query not event(DataDecryption1(x_39))
RESULT not event(DataDecryption1(x_39)) is true.
-- Query not event(DataDecryption2(x_40))
Completing...
Starting query not event(DataDecryption2(x_40))
RESULT not event(DataDecryption2(x_40)) is true.
-- Query event(DataDecryption1(x_41)) ==> event(DataDecryption2(x_41))
Completing...
Starting query event(DataDecryption1(x_41)) ==> event(DataDecryption2(x_41
   ))
RESULT event(DataDecryption1(x_41)) ==> event(DataDecryption2(x_41)) is
   true.
-- Query inj-event(DataDecryption1(x_42)) ==> inj-event(DataDecryption2(
   x_42))
Completing...
Starting query inj-event(DataDecryption1(x_42)) ==> inj-event(
   DataDecryption2(x_42))
RESULT inj-event(DataDecryption1(x_42)) ==> inj-event(DataDecryption2(x_42
   )) is true.
```

Appendix 11 – PSK and EAP-PSK Authentication

EAP-PSK and PSK authentication protocols are similar but not that different. Their biggest differences lie in the transport field and key lengths, but authentication flow is the same as the goal.
EAP-PSK is composed of four messages: [148]
First message sent by the server to the peer which starts the mutual authentication procedure and consists of a random value chosen by the server.
Second message sent by the peer to the server which contains a random value chosen by the peer and an authentication tag over both random values as well as the peer and server's permanent network access identifier (NAI), that proves the identity of the peer to the server.
Third message sent by the server to the peer that contains an authentication tag calculated over the random value chosen by the peer and the server's permanent full NAI that proves the identity of the server to the peer. This message may also contain data encapsulated in a protected channel that has just been set up as a result of the authentication procedure.
Fourth message sent by the peer to the server that may also contain data encapsulated in a protected channel that has just been set up as a result of the authentication procedure

ProVerif Verification

```
set traceDisplay = long.

query attacker(PSK).
query attacker(MasterKey).
query attacker(DecryptedData).
query attacker(DecryptedData2).

query x:bitstring; event(DataAESDecryption1(x)).
query x:bitstring; event(DataAESDecryption2(x)).
query x:bitstring; event(DataAESDecryption1(x)) ==> event(
   DataAESDecryption2(x)).
query x:bitstring; inj-event(DataAESDecryption1(x)) ==> inj-event(
   DataAESDecryption2(x)).


type challenge.
type response.
type nonce.
type key.

free c:channel.

free RequestIdentity:bitstring[private].
free ResponseIdentity:bitstring[private].
free Challenge1:bitstring[private].
free PSKChallenge:challenge[private].
free MasterKey:key[private].
free PSK:key [private].
free PSKChallengeResponse:response[private].
free TestHello:bitstring[private].
free TestHelloAck:bitstring[private].
free Success:bitstring[private].
free EncryptedData:nonce [private].
free EncryptedData2:nonce[private].
free DecryptedData:bitstring[private].
free DecryptedData2:bitstring[private].
free DBResponse:bitstring[private].

fun PSKChallengeGeneration(bitstring, key):challenge.
fun SolveChallenge(challenge, key):response.
fun AESEncryption(bitstring,key):nonce.
reduc forall x:bitstring, y:key;AESDecryption(AESEncryption(x,y),y)=x.
event ResponseCheck(bitstring).
event ChallengeGeneration(challenge).
event DataAESDecryption1(bitstring).
event DataAESDecryption2(bitstring).
event ChallengeSolving(response).

let Server1 =
  out(c, RequestIdentity).
let Client1 =
  in(c, RequestIdentity:bitstring);
  out(c, ResponseIdentity).
let Server2 =
  in(c, ResponseIdentity:bitstring);
  let ResponseIdentity = DBResponse in
    event ResponseCheck(ResponseIdentity);
  if ResponseIdentity = DBResponse then
  (
    new PSKChallenge:challenge;
    new Challenge1:bitstring;
    let PSKChallenge = PSKChallengeGeneration(Challenge1, MasterKey) in
      event ChallengeGeneration(PSKChallenge);
    out(c, PSKChallenge)
    )
    else
    (
      0
      ).
let Client2 =
  in(c, PSKChallenge:challenge);
  let PSKChallengeResponse = SolveChallenge(PSKChallenge, PSK) in
    event ChallengeSolving(PSKChallengeResponse);
  out(c, PSKChallengeResponse).

let Server3 =
  in(c, PSKChallengeResponse:response);
  if PSKChallengeResponse = SolveChallenge(PSKChallenge, MasterKey) then
  (
    out(c, AESEncryption(TestHello, MasterKey))
    )
    else
    (
      0
      ).

let Client3 =
  in(c, EncryptedData:nonce);
  let DecryptedData = AESDecryption(EncryptedData, PSK) in
    event DataAESDecryption1(DecryptedData);
  out(c, AESEncryption(TestHelloAck, PSK)).

let Server4 =
  in(c, EncryptedData2:nonce);
  let DecryptedData2 = AESDecryption(EncryptedData2, PSK) in
    event DataAESDecryption2(DecryptedData2).

process (
  !Server1|!Client1|!Server2|!Client2|!Server3|!Client3|!Server4
  )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, RequestIdentity)
) | (
    {3}!
    {4}in(c, RequestIdentity_16: bitstring);
    {5}out(c, ResponseIdentity)
) | (
    {6}!
    {7}in(c, ResponseIdentity_17: bitstring);
    {8}let ResponseIdentity_18: bitstring = DBResponse in
    {9}event ResponseCheck(ResponseIdentity_18);
    {10}if (ResponseIdentity_18 = DBResponse) then
    {11}new PSKChallenge_19: challenge;
    {12}new Challenge1_20: bitstring;
    {13}let PSKChallenge_21: challenge = PSKChallengeGeneration(
       Challenge1_20,MasterKey) in
    {14}event ChallengeGeneration(PSKChallenge_21);
    {15}out(c, PSKChallenge_21)
) | (
    {16}!
    {17}in(c, PSKChallenge_22: challenge);
    {18}let PSKChallengeResponse_23: response = SolveChallenge(
       PSKChallenge_22,PSK) in
    {19}event ChallengeSolving(PSKChallengeResponse_23);
    {20}out(c, PSKChallengeResponse_23)
) | (
    {21}!
    {22}in(c, PSKChallengeResponse_24: response);
    {23}if (PSKChallengeResponse_24 = SolveChallenge(PSKChallenge,
       MasterKey)) then
    {24}out(c, AESEncryption(TestHello,MasterKey))
) | (
    {25}!
    {26}in(c, EncryptedData_25: nonce);
    {27}let DecryptedData_26: bitstring = AESDecryption(EncryptedData_25,
       PSK) in
    {28}event DataAESDecryption1(DecryptedData_26);
    {29}out(c, AESEncryption(TestHelloAck,PSK))
) | (
    {30}!
    {31}in(c, EncryptedData2_27: nonce);
    {32}let DecryptedData2_28: bitstring = AESDecryption(EncryptedData2_27
       ,PSK) in
    {33}event DataAESDecryption2(DecryptedData2_28)
)

-- Query not attacker(PSK[])
Completing...
Starting query not attacker(PSK[])
RESULT not attacker(PSK[]) is true.
-- Query not attacker(MasterKey[])
Completing...
Starting query not attacker(MasterKey[])
RESULT not attacker(MasterKey[]) is true.
-- Query not attacker(DecryptedData[])
Completing...
Starting query not attacker(DecryptedData[])
RESULT not attacker(DecryptedData[]) is true.
-- Query not attacker(DecryptedData2[])
Completing...
Starting query not attacker(DecryptedData2[])
RESULT not attacker(DecryptedData2[]) is true.
-- Query not event(DataAESDecryption1(x_29))
Completing...
Starting query not event(DataAESDecryption1(x_29))
RESULT not event(DataAESDecryption1(x_29)) is true.
-- Query not event(DataAESDecryption2(x_30))
Completing...
Starting query not event(DataAESDecryption2(x_30))
RESULT not event(DataAESDecryption2(x_30)) is true.
-- Query event(DataAESDecryption1(x_31)) ==> event(DataAESDecryption2(x_31
   ))
Completing...
Starting query event(DataAESDecryption1(x_31)) ==> event(
   DataAESDecryption2(x_31))
RESULT event(DataAESDecryption1(x_31)) ==> event(DataAESDecryption2(x_31))
     is true.
-- Query inj-event(DataAESDecryption1(x_32)) ==> inj-event(
   DataAESDecryption2(x_32))
Completing...
```

```
Starting query inj-event(DataAESDecryption1(x_32)) ==> inj-event(
    DataAESDecryption2(x_32))
RESULT inj-event(DataAESDecryption1(x_32)) ==> inj-event(
    DataAESDecryption2(x_32)) is true.
```

Appendix 12 – EAP-TLS Authentication

EAP-TLS is based on TLS, which is used to provide protected cipher-suite negotiation, mutual authentication, and key management. After the EAP-TLS negotiation is completed, the two end-points can securely communicate within the encrypted TLS tunnel. Therefore, user's identity and password will not be revealed. Because TLS provides a way to use certificates for both user and server to authenticate each other, a user, in addition to being authenticated, can also authenticate the network. [89]

EAP-TLS uses the TLS public key certificate authentication mechanism within EAP to provide mutual authentication of client to server and server to client. With EAP-TLS, both the client and the server must be assigned a digital certificate signed by a Certificate Authority (CA) that they both trust.
EAP-TLS provides :
Mutual authentication
Key exchange
Fragmentation and reassembly
Fast reconnect.
Diffie-Hellman authentication has been described in Appendix 3 and TLS Appendix 8.

ProVerif Verification

```
set traceDisplay = long.

query attacker(EncryptionKey).
query attacker(Password).
query attacker(ServerKey).
query attacker(DecryptedTraffic).
query attacker(DecryptedTraffic2).


query x:bitstring; event(DataDecryption1(x)).
query x:bitstring; event(DataDecryption2(x)).
query x:bitstring; event(DataDecryption1(x)) ==> event(DataDecryption2(x))
    .
query x:bitstring; inj-event(DataDecryption1(x)) ==> inj-event(
    DataDecryption2(x)).


type certificate.
type password.
type challenge.
type key.
type response.
type nonce.
type material.

free c:channel.
free AccessChallenge:challenge[private].
free EAPRequest,DecryptedTraffic2, DecryptedTraffic:bitstring[private].
free EAPResponseIdentity:bitstring[private].
free EAPTLSStart,TLSHello2, TLSHello2ACK, ClientKeyExchange, Cipher,
    ClientTLSFinished:bitstring[private].
free EAPTLSHello, EAPTLSServerHello, Handshake, ServerKeyExchange,
    CertificateRequest, ServerHelloDone:bitstring[private].
free ServerCertificate, ClientCertificate, ValidCA:certificate[private].
free Password:password[private].
free EncKey:material[private].
free EncryptionKey:key[private].
free ServerKey:key[private].
free Hello:bitstring[private].
free HelloACK:bitstring[private].
```

```
fun SolveChallenge(challenge, password):response.
fun EncryptionKeyGeneration(material):key.

fun TrafficEncryption(bitstring, key):nonce.
reduc forall x:bitstring, y:key; TrafficDecryption(TrafficEncryption(x,y),
   y)=x.

event ServerCertificateValidation(certificate).
event ChallengeSolving(response).
event EncryptionKeyGen(key).
event ServerKeyTrafficDecryption(bitstring).
event ClientKeyTrafficDecryption(bitstring).
event ClientCertificateValidation(certificate).
event DataDecryption1(bitstring).
event DataDecryption2(bitstring).

let AP1 =
  out(c, EAPRequest).
let Client1 =
  in(c, EAPRequest:bitstring);
  out(c, EAPResponseIdentity).
let AP2 =
  in(c, EAPResponseIdentity:bitstring);
  out(c, EAPResponseIdentity).
let Server1 =
  in(c, EAPResponseIdentity:bitstring);
  out(c, EAPTLSStart).
let AP3 =
  in(c, EAPTLStart:bitstring);
  out(c, EAPTLSStart).
let Client2 =
  in(c, EAPTLSStart:bitstring);
  out(c, EAPTLSHello).
let AP4 =
  in(c, EAPTLSHello:bitstring);
  out(c, EAPTLSHello).
let Server2 =
  in(c, EAPTLSHello:bitstring);
  new ServerCertificate:certificate;
  out(c, (EAPTLSServerHello, (Handshake, (ServerCertificate, (
     ServerKeyExchange, (CertificateRequest, ServerHelloDone)))))).
```

```
let AP5 =
  in(c, (EAPTLSServerHello:bitstring, (Handshake:bitstring, (
      ServerCertificate:certificate, (ServerKeyExchange:bitstring, (
      CertificateRequest:bitstring, ServerHelloDone:bitstring))))));
  out(c, (EAPTLSServerHello, (Handshake, (ServerCertificate, (
      ServerKeyExchange, (CertificateRequest, ServerHelloDone)))))).

let Client3 =
  in(c, (EAPTLSServerHello:bitstring, (Handshake:bitstring, (
      ServerCertificate:certificate, (ServerKeyExchange:bitstring, (
      CertificateRequest:bitstring, ServerHelloDone:bitstring))))));
  let ServerCertificate = ValidCA in
    event ServerCertificateValidation(ServerCertificate);
  if ServerCertificate = ValidCA then
  (
    new ClientCertificate:certificate;
    out(c, (ClientCertificate, (ClientKeyExchange, (Cipher,
        ClientTLSFinished))))
    )
    else
    (
      0
      ).
let AP6 =
  in(c, (ClientCertificate:certificate, (ClientKeyExchange:bitstring, (
      Cipher:bitstring, ClientTLSFinished:bitstring))));
  out(c, (ClientCertificate, (ClientKeyExchange, (Cipher,
      ClientTLSFinished)))).

let Server3 =
  in(c, (ClientCertificate:certificate, (ClientKeyExchange:bitstring, (
      Cipher:bitstring, ClientTLSFinished:bitstring))));
  let ClientCertificate = ValidCA in
    event ClientCertificateValidation(ClientCertificate);
  if ClientCertificate = ValidCA then
  (
    out(c, AccessChallenge)
    )
    else
    (
      0
      ).
```

```
let AP7 =
  in(c, AccessChallenge:challenge);
  out(c, AccessChallenge).

let Client4 =
  in(c, AccessChallenge:challenge);
  let SolvedChallenge = SolveChallenge(AccessChallenge, Password) in
    event ChallengeSolving(SolvedChallenge);
  out(c, SolvedChallenge).

let AP8 =
  in(c, SolvedChallenge:response);
  out(c, SolvedChallenge).

let Server4 =
  in(c, SolvedChallenge:response);
  if SolvedChallenge = SolveChallenge(AccessChallenge, Password) then
  (
    new EncKey:material;
    out(c, EncKey)
    )
    else
    (
      0
      ).

let AP9 =
  in(c, EncKey:material);
  out(c, EncKey).

let Client5 =
  in(c, EncKey:material);
  let EncryptionKey = EncryptionKeyGeneration(EncKey) in
    event EncryptionKeyGen(EncryptionKey);
  out(c, TrafficEncryption(TLSHello2, EncryptionKey)).

let AP10 =
  in(c, EncryptedTraffic:nonce);
  out(c, EncryptedTraffic).

let Server5 =
  in(c, EncryptedTraffic:nonce);
  let DecryptedTraffic = TrafficDecryption(EncryptedTraffic, ServerKey) in
    event ServerKeyTrafficDecryption(DecryptedTraffic);
  out(c, TrafficEncryption(TLSHello2ACK, ServerKey)).

let AP11 =
  in(c, EncryptedTraffic2:nonce);
  out(c, EncryptedTraffic2).

let Client6 =
  in(c, EncryptedTraffic2:nonce);
  let DecryptedTraffic2 = TrafficDecryption(EncryptedTraffic2,
     EncryptionKey) in
    event ClientKeyTrafficDecryption(DecryptedTraffic2).

    process (
      !AP1|!Client1|!AP2|!Server1|!AP3|!Client2|!AP4|!Server2|!AP5|!
         Client3|!AP6|!Server3|!AP7|!Client4|!AP8|!Server4|!AP9|!Client5|!
         AP10|!Server5|!AP11|!Client6
      )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, EAPRequest)
) | (
    {3}!
    {4}in(c, EAPRequest_16: bitstring);
    {5}out(c, EAPResponseIdentity)
) | (
    {6}!
    {7}in(c, EAPResponseIdentity_17: bitstring);
    {8}out(c, EAPResponseIdentity_17)
) | (
    {9}!
    {10}in(c, EAPResponseIdentity_18: bitstring);
    {11}out(c, EAPTLSStart)
) | (
    {12}!
    {13}in(c, EAPTLStart: bitstring);
    {14}out(c, EAPTLSStart)
) | (
    {15}!
    {16}in(c, EAPTLSStart_19: bitstring);
    {17}out(c, EAPTLSHello)
) | (
    {18}!
    {19}in(c, EAPTLSHello_20: bitstring);
    {20}out(c, EAPTLSHello_20)
) | (
    {21}!
    {22}in(c, EAPTLSHello_21: bitstring);
    {23}new ServerCertificate_22: certificate;
    {24}out(c, (EAPTLSServerHello,(Handshake,(ServerCertificate_22,(
       ServerKeyExchange,(CertificateRequest,ServerHelloDone))))))
) | (
    {25}!
    {26}in(c, (EAPTLSServerHello_23: bitstring,(Handshake_24: bitstring,(
       ServerCertificate_25: certificate,(ServerKeyExchange_26: bitstring
       ,(CertificateRequest_27: bitstring,ServerHelloDone_28: bitstring)))
       )));
    {27}out(c, (EAPTLSServerHello_23,(Handshake_24,(ServerCertificate_25,(
        ServerKeyExchange_26,(CertificateRequest_27,ServerHelloDone_28)))))
        )
) | (
    {28}!
    {29}in(c, (EAPTLSServerHello_29: bitstring,(Handshake_30: bitstring,(
        ServerCertificate_31: certificate,(ServerKeyExchange_32: bitstring
        ,(CertificateRequest_33: bitstring,ServerHelloDone_34: bitstring)))
        )));
    {30}let ServerCertificate_35: certificate = ValidCA in
    {31}event ServerCertificateValidation(ServerCertificate_35);
    {32}if (ServerCertificate_35 = ValidCA) then
    {33}new ClientCertificate_36: certificate;
    {34}out(c, (ClientCertificate_36,(ClientKeyExchange,(Cipher,
        ClientTLSFinished))))
) | (
    {35}!
    {36}in(c, (ClientCertificate_37: certificate,(ClientKeyExchange_38:
        bitstring,(Cipher_39: bitstring,ClientTLSFinished_40: bitstring))))
        ;
    {37}out(c, (ClientCertificate_37,(ClientKeyExchange_38,(Cipher_39,
        ClientTLSFinished_40))))
) | (
    {38}!
    {39}in(c, (ClientCertificate_41: certificate,(ClientKeyExchange_42:
        bitstring,(Cipher_43: bitstring,ClientTLSFinished_44: bitstring))))
        ;
    {40}let ClientCertificate_45: certificate = ValidCA in
    {41}event ClientCertificateValidation(ClientCertificate_45);
    {42}if (ClientCertificate_45 = ValidCA) then
    {43}out(c, AccessChallenge)
) | (
    {44}!
    {45}in(c, AccessChallenge_46: challenge);
    {46}out(c, AccessChallenge_46)
) | (
    {47}!
    {48}in(c, AccessChallenge_47: challenge);
    {49}let SolvedChallenge: response = SolveChallenge(AccessChallenge_47,
        Password) in
    {50}event ChallengeSolving(SolvedChallenge);
    {51}out(c, SolvedChallenge)
```

```
) | (
    {52}!
    {53}in(c, SolvedChallenge_48: response);
    {54}out(c, SolvedChallenge_48)
) | (
    {55}!
    {56}in(c, SolvedChallenge_49: response);
    {57}if (SolvedChallenge_49 = SolveChallenge(AccessChallenge,Password))
        then
    {58}new EncKey_50: material;
    {59}out(c, EncKey_50)
) | (
    {60}!
    {61}in(c, EncKey_51: material);
    {62}out(c, EncKey_51)
) | (
    {63}!
    {64}in(c, EncKey_52: material);
    {65}let EncryptionKey_53: key = EncryptionKeyGeneration(EncKey_52) in
    {66}event EncryptionKeyGen(EncryptionKey_53);
    {67}out(c, TrafficEncryption(TLSHello2,EncryptionKey_53))
) | (
    {68}!
    {69}in(c, EncryptedTraffic: nonce);
    {70}out(c, EncryptedTraffic)
) | (
    {71}!
    {72}in(c, EncryptedTraffic_54: nonce);
    {73}let DecryptedTraffic_55: bitstring = TrafficDecryption(
       EncryptedTraffic_54,ServerKey) in
    {74}event ServerKeyTrafficDecryption(DecryptedTraffic_55);
    {75}out(c, TrafficEncryption(TLSHello2ACK,ServerKey))
) | (
    {76}!
    {77}in(c, EncryptedTraffic2: nonce);
    {78}out(c, EncryptedTraffic2)
) | (
    {79}!
    {80}in(c, EncryptedTraffic2_56: nonce);
    {81}let DecryptedTraffic2_57: bitstring = TrafficDecryption(
       EncryptedTraffic2_56,EncryptionKey) in
    {82}event ClientKeyTrafficDecryption(DecryptedTraffic2_57)
```

```
)

-- Query not attacker(EncryptionKey[])
Completing...
Starting query not attacker(EncryptionKey[])
RESULT not attacker(EncryptionKey[]) is true.
-- Query not attacker(Password[])
Completing...
Starting query not attacker(Password[])
RESULT not attacker(Password[]) is true.
-- Query not attacker(ServerKey[])
Completing...
Starting query not attacker(ServerKey[])
RESULT not attacker(ServerKey[]) is true.
-- Query not attacker(DecryptedTraffic[])
Completing...
Starting query not attacker(DecryptedTraffic[])
RESULT not attacker(DecryptedTraffic[]) is true.
-- Query not attacker(DecryptedTraffic2[])
Completing...
Starting query not attacker(DecryptedTraffic2[])
RESULT not attacker(DecryptedTraffic2[]) is true.
-- Query not event(DataDecryption1(x_58))
Completing...
Starting query not event(DataDecryption1(x_58))
RESULT not event(DataDecryption1(x_58)) is true.
-- Query not event(DataDecryption2(x_59))
Completing...
Starting query not event(DataDecryption2(x_59))
RESULT not event(DataDecryption2(x_59)) is true.
-- Query event(DataDecryption1(x_60)) ==> event(DataDecryption2(x_60))
Completing...
Starting query event(DataDecryption1(x_60)) ==> event(DataDecryption2(x_60
    ))
RESULT event(DataDecryption1(x_60)) ==> event(DataDecryption2(x_60)) is
    true.
-- Query inj-event(DataDecryption1(x_61)) ==> inj-event(DataDecryption2(
    x_61))
Completing...
Starting query inj-event(DataDecryption1(x_61)) ==> inj-event(
    DataDecryption2(x_61))
RESULT inj-event(DataDecryption1(x_61)) ==> inj-event(DataDecryption2(x_61
)) is true.
```

Appendix 13 – EAP-TTLS Authentication

EAP-TTLS extends EAP-TLS to exchange information between client and server by using the secure tunnel established by TLS negotiation. An EAP-TTLS negotiation has two phases: the TLS handshake phase and the TLS tunnel phase. During phase one, TLS is used for the client to authenticate the server. Similarly as in EAP-TLS, the authentication is done by using certificates. A secure TLS tunnel is established after the phase-one handshake. In phase two, the secure TLS tunnel can be used for other information exchanges, such as additional user authentication key, communication of accounting information, and so forth. [2]

It is very similar to the TLS authentication, but the outcome is different, which is a secure tunnel to the destination, which is used for authentication, where EAP-TLS has already authenticated and uses tunnel for communication.

Diffie-Hellman authentication has been described in Appendix 3 and TLS Appendix 8 and EAP-TLS Appendix 12.

ProVerif Verification

```
set traceDisplay = long.

query attacker(ClientKey).
query attacker(ServerKey).
query attacker(DiameterPassword).
query attacker(ServerMKey).
query attacker(ClientMKey).

query attacker(DecryptedTLSHello).
query attacker(DecryptedTLSHelloACK).


query x:bitstring; event(TLSHelloDecryption(x)).
query x:bitstring; event(TLSHelloACKDecryption(x)).
query x:bitstring; event(TLSHelloDecryption(x)) ==> event(
   TLSHelloACKDecryption(x)).
query x:bitstring; inj-event(TLSHelloDecryption(x)) ==> inj-event(
   TLSHelloACKDecryption(x)).


type certificate.
type password.
type challenge.
type key.
type response.
type nonce.
type material.

free c:channel.
free AccessChallenge:challenge[private].
free EAPRequest,DecryptedTraffic2, DecryptedTraffic:bitstring[private].
free EAPResponseIdentity, DecryptedTLSHello, DecryptedTLSHelloACK:
   bitstring[private].
free EAPTLSStart,TLSHello2, TLSHello2ACK, ClientKeyExchange, Cipher,
   ClientTLSFinished, TLSHello2ACKACK:bitstring[private].
free EAPTLSHello, EAPTLSServerHello, Handshake, ServerKeyExchange,
   CertificateRequest, ServerHelloDone:bitstring[private].
free ServerCertificate, ClientCertificate, ValidCA:certificate[private].
free DiameterPassword:password[private].
free ClientKey:key[private].
free ServerKey:key[private].
free EncKey:material [private].
free ClientPubKey:key[private].
free ServerPubKey:key[private].

type mkey.
free ServerMKey:mkey [private].
free ClientMKey:mkey [private].
fun MKeyGeneration(bitstring, key, key):mkey.
event ClientMKeyGeneration(mkey).
event ServerMKeyGeneration(mkey).

fun SolveChallenge(challenge, password):response.
fun EncryptionKeyGeneration(material):key.

fun TrafficEncryption(bitstring, mkey):nonce.
reduc forall x:bitstring, y:mkey; TrafficDecryption(TrafficEncryption(x,y)
   ,y)=x.

event ServerCertificateValidation(certificate).
event ChallengeSolving(response).
event EncryptionKeyGen(key).
event ServerKeyTrafficDecryption(bitstring).
event ClientKeyTrafficDecryption(bitstring).
event ClientCertificateValidation(certificate).
event SecretKeyDecryptionBitstring(bitstring).
event TLSHelloDecryption(bitstring).
event TLSHelloACKDecryption(bitstring).

let AP1 =
  out(c, EAPRequest).

let Client1 =
  in(c, EAPRequest:bitstring);
  out(c, EAPResponseIdentity).
let AP2 =
  in(c, EAPResponseIdentity:bitstring);
  out(c, EAPResponseIdentity).
let Server1 =
  in(c, EAPResponseIdentity:bitstring);
  out(c, EAPTLSStart).
let AP3 =
  in(c, EAPTLStart:bitstring);
  out(c, EAPTLSStart).
let Client2 =
  in(c, EAPTLSStart:bitstring);
  out(c, EAPTLSHello).
let AP4 =
  in(c, EAPTLSHello:bitstring);
  out(c, EAPTLSHello).
let Server2 =
  in(c, EAPTLSHello:bitstring);
  new ServerCertificate:certificate;
  out(c, (EAPTLSServerHello, (Handshake, (ServerCertificate, (
     ServerKeyExchange, (CertificateRequest, ServerHelloDone)))))).

let AP5 =
  in(c, (EAPTLSServerHello:bitstring, (Handshake:bitstring, (
     ServerCertificate:certificate, (ServerKeyExchange:bitstring, (
     CertificateRequest:bitstring, ServerHelloDone:bitstring))))));
  out(c, (EAPTLSServerHello, (Handshake, (ServerCertificate, (
     ServerKeyExchange, (CertificateRequest, ServerHelloDone)))))).

let Client3 =
  in(c, (EAPTLSServerHello:bitstring, (Handshake:bitstring, (
     ServerCertificate:certificate, (ServerKeyExchange:bitstring, (
     CertificateRequest:bitstring, ServerHelloDone:bitstring))))));
  let ServerCertificate = ValidCA in
    event ServerCertificateValidation(ServerCertificate);
  if ServerCertificate = ValidCA then
  (
    new ClientCertificate:certificate;
    out(c, (ClientCertificate, (ClientKeyExchange, (Cipher,
       ClientTLSFinished))))
    )
    else
    (
      0
      ).
let AP6 =
  in(c, (ClientCertificate:certificate, (ClientKeyExchange:bitstring, (
     Cipher:bitstring, ClientTLSFinished:bitstring))));
  out(c, (ClientCertificate, (ClientKeyExchange, (Cipher,
     ClientTLSFinished)))).
```

```
let Server3 =
  in(c, (ClientCertificate:certificate, (ClientKeyExchange:bitstring, (
     Cipher:bitstring, ClientTLSFinished:bitstring))));
  let ClientCertificate = ValidCA in
    event ClientCertificateValidation(ClientCertificate);
  if ClientCertificate = ValidCA then
  (
    out(c, AccessChallenge)
    )
    else
    (
      0
      ).

let AP7 =
  in(c, AccessChallenge:challenge);
  out(c, AccessChallenge).

let Client4 =
  in(c, AccessChallenge:challenge);
  let SolvedChallenge = SolveChallenge(AccessChallenge, DiameterPassword)
     in
    event ChallengeSolving(SolvedChallenge);
  out(c, SolvedChallenge).

let AP8 =
  in(c, SolvedChallenge:response);
  out(c, SolvedChallenge).

let Server4 =
  in(c, SolvedChallenge:response);
  if SolvedChallenge = SolveChallenge(AccessChallenge, DiameterPassword)
     then
  (
    let ServerMKey = MKeyGeneration(EAPTLSHello, ClientPubKey, ServerKey)
       in
      event ServerMKeyGeneration(ServerMKey);
    out(c, (EncKey, TrafficEncryption(TLSHello2, ServerMKey)))
    )
    else
    (
      0
      ).

let Client5 =
  in(c, (EncKey:material, EncryptedTraffic:nonce));
  let ClientKey = EncryptionKeyGeneration(EncKey) in
    event EncryptionKeyGen(ClientKey);
  let ClientMKey = MKeyGeneration(EAPTLSServerHello, ServerPubKey,
      ClientKey) in
    event ClientMKeyGeneration(ClientMKey);
  let DecryptedTLSHello = TrafficDecryption(EncryptedTraffic, ClientMKey)
      in
    event TLSHelloDecryption(DecryptedTLSHello);
  out(c, TrafficEncryption(TLSHello2ACK, ClientMKey)).

let Server5 =
  in(c, EncryptedTraffic2:nonce);
  let DecryptedTLSHelloACK = TrafficDecryption(EncryptedTraffic2,
      ServerMKey) in
    event TLSHelloACKDecryption(DecryptedTLSHelloACK).


process (
  !AP1|!Client1|!AP2|!Server1|!AP3|!Client2|!AP4|!Server2|!AP5|!Client3|!
      AP6|!Server3|!AP7|!Client4|!AP8|!Server4|!Client5|!Server5
  )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, EAPRequest)
) | (
    {3}!
    {4}in(c, EAPRequest_16: bitstring);
    {5}out(c, EAPResponseIdentity)
) | (
    {6}!
    {7}in(c, EAPResponseIdentity_17: bitstring);
    {8}out(c, EAPResponseIdentity_17)
) | (
    {9}!
    {10}in(c, EAPResponseIdentity_18: bitstring);
    {11}out(c, EAPTLSStart)
) | (
    {12}!
    {13}in(c, EAPTLStart: bitstring);
    {14}out(c, EAPTLSStart)
) | (
    {15}!
    {16}in(c, EAPTLSStart_19: bitstring);
    {17}out(c, EAPTLSHello)
) | (
    {18}!
    {19}in(c, EAPTLSHello_20: bitstring);
    {20}out(c, EAPTLSHello_20)
) | (
    {21}!
    {22}in(c, EAPTLSHello_21: bitstring);
    {23}new ServerCertificate_22: certificate;
    {24}out(c, (EAPTLSServerHello,(Handshake,(ServerCertificate_22,(
       ServerKeyExchange,(CertificateRequest,ServerHelloDone))))))
) | (
    {25}!
    {26}in(c, (EAPTLSServerHello_23: bitstring,(Handshake_24: bitstring,(
       ServerCertificate_25: certificate,(ServerKeyExchange_26: bitstring
       ,(CertificateRequest_27: bitstring,ServerHelloDone_28: bitstring)))
       )));
    {27}out(c, (EAPTLSServerHello_23,(Handshake_24,(ServerCertificate_25,(
       ServerKeyExchange_26,(CertificateRequest_27,ServerHelloDone_28)))))
       )
) | (
    {28}!
    {29}in(c, (EAPTLSServerHello_29: bitstring,(Handshake_30: bitstring,(
       ServerCertificate_31: certificate,(ServerKeyExchange_32: bitstring
       ,(CertificateRequest_33: bitstring,ServerHelloDone_34: bitstring)))
       )));
    {30}let ServerCertificate_35: certificate = ValidCA in
    {31}event ServerCertificateValidation(ServerCertificate_35);
    {32}if (ServerCertificate_35 = ValidCA) then
    {33}new ClientCertificate_36: certificate;
    {34}out(c, (ClientCertificate_36,(ClientKeyExchange,(Cipher,
       ClientTLSFinished))))
) | (
    {35}!
    {36}in(c, (ClientCertificate_37: certificate,(ClientKeyExchange_38:
       bitstring,(Cipher_39: bitstring,ClientTLSFinished_40: bitstring))))
       ;
    {37}out(c, (ClientCertificate_37,(ClientKeyExchange_38,(Cipher_39,
       ClientTLSFinished_40))))
) | (
    {38}!
    {39}in(c, (ClientCertificate_41: certificate,(ClientKeyExchange_42:
       bitstring,(Cipher_43: bitstring,ClientTLSFinished_44: bitstring))))
       ;
    {40}let ClientCertificate_45: certificate = ValidCA in
    {41}event ClientCertificateValidation(ClientCertificate_45);
    {42}if (ClientCertificate_45 = ValidCA) then
    {43}out(c, AccessChallenge)
) | (
    {44}!
    {45}in(c, AccessChallenge_46: challenge);
    {46}out(c, AccessChallenge_46)
) | (
    {47}!
    {48}in(c, AccessChallenge_47: challenge);
    {49}let SolvedChallenge: response = SolveChallenge(AccessChallenge_47,
       DiameterPassword) in
    {50}event ChallengeSolving(SolvedChallenge);
    {51}out(c, SolvedChallenge)
```

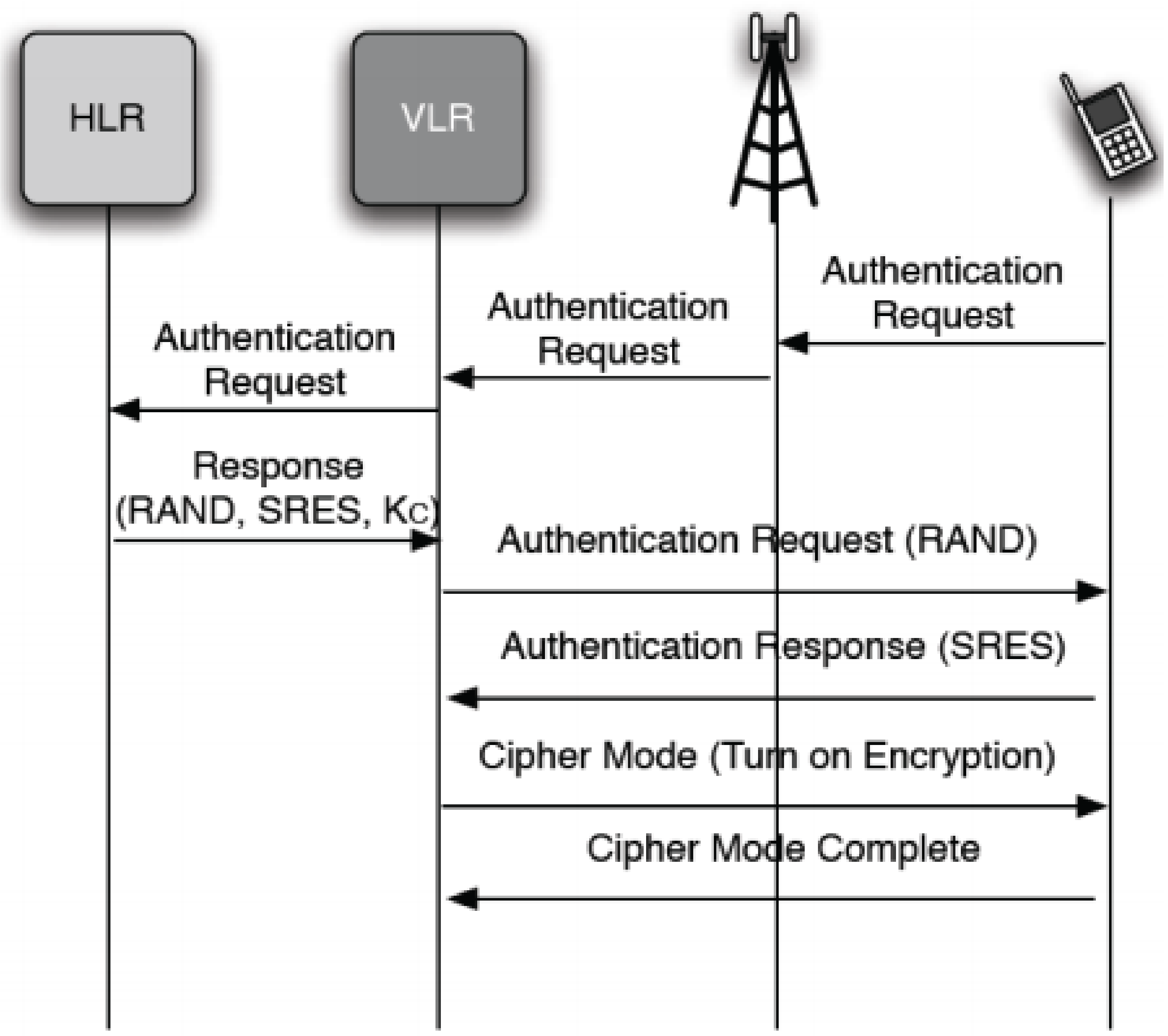
```
) | (
    {52}!
    {53}in(c, SolvedChallenge_48: response);
    {54}out(c, SolvedChallenge_48)
) | (
    {55}!
    {56}in(c, SolvedChallenge_49: response);
    {57}if (SolvedChallenge_49 = SolveChallenge(AccessChallenge,
       DiameterPassword)) then
    {58}let ServerMKey_50: mkey = MKeyGeneration(EAPTLSHello,ClientPubKey,
       ServerKey) in
    {59}event ServerMKeyGeneration(ServerMKey_50);
    {60}out(c, (EncKey,TrafficEncryption(TLSHello2,ServerMKey_50)))
) | (
    {61}!
    {62}in(c, (EncKey_51: material,EncryptedTraffic: nonce));
    {63}let ClientKey_52: key = EncryptionKeyGeneration(EncKey_51) in
    {64}event EncryptionKeyGen(ClientKey_52);
    {65}let ClientMKey_53: mkey = MKeyGeneration(EAPTLSServerHello,
       ServerPubKey,ClientKey_52) in
    {66}event ClientMKeyGeneration(ClientMKey_53);
    {67}let DecryptedTLSHello_54: bitstring = TrafficDecryption(
       EncryptedTraffic,ClientMKey_53) in
    {68}event TLSHelloDecryption(DecryptedTLSHello_54);
    {69}out(c, TrafficEncryption(TLSHello2ACK,ClientMKey_53))
) | (
    {70}!
    {71}in(c, EncryptedTraffic2: nonce);
    {72}let DecryptedTLSHelloACK_55: bitstring = TrafficDecryption(
       EncryptedTraffic2,ServerMKey) in
    {73}event TLSHelloACKDecryption(DecryptedTLSHelloACK_55)
)

-- Query not attacker(ClientKey[])
Completing...
Starting query not attacker(ClientKey[])
RESULT not attacker(ClientKey[]) is true.
-- Query not attacker(ServerKey[])
Completing...
Starting query not attacker(ServerKey[])
RESULT not attacker(ServerKey[]) is true.
-- Query not attacker(DiameterPassword[])
```

```
Completing...
Starting query not attacker(DiameterPassword[])
RESULT not attacker(DiameterPassword[]) is true.
-- Query not attacker(ServerMKey[])
Completing...
Starting query not attacker(ServerMKey[])
RESULT not attacker(ServerMKey[]) is true.
-- Query not attacker(ClientMKey[])
Completing...
Starting query not attacker(ClientMKey[])
RESULT not attacker(ClientMKey[]) is true.
-- Query not attacker(DecryptedTLSHello[])
Completing...
Starting query not attacker(DecryptedTLSHello[])
RESULT not attacker(DecryptedTLSHello[]) is true.
-- Query not attacker(DecryptedTLSHelloACK[])
Completing...
Starting query not attacker(DecryptedTLSHelloACK[])
RESULT not attacker(DecryptedTLSHelloACK[]) is true.
-- Query not event(TLSHelloDecryption(x_56))
Completing...
Starting query not event(TLSHelloDecryption(x_56))
RESULT not event(TLSHelloDecryption(x_56)) is true.
-- Query not event(TLSHelloACKDecryption(x_57))
Completing...
Starting query not event(TLSHelloACKDecryption(x_57))
RESULT not event(TLSHelloACKDecryption(x_57)) is true.
-- Query event(TLSHelloDecryption(x_58)) ==> event(TLSHelloACKDecryption(
   x_58))
Completing...
Starting query event(TLSHelloDecryption(x_58)) ==> event(
   TLSHelloACKDecryption(x_58))
RESULT event(TLSHelloDecryption(x_58)) ==> event(TLSHelloACKDecryption(
   x_58)) is true.
-- Query inj-event(TLSHelloDecryption(x_59)) ==> inj-event(
   TLSHelloACKDecryption(x_59))
Completing...
Starting query inj-event(TLSHelloDecryption(x_59)) ==> inj-event(
   TLSHelloACKDecryption(x_59))
RESULT inj-event(TLSHelloDecryption(x_59)) ==> inj-event(
   TLSHelloACKDecryption(x_59)) is true.
```

Appendix 14 – GSM Authentication

The GSM network authenticates the identity of the subscriber through the use of a challenge-response mechanism. A 128-bit Random Number (RAND) is sent to the Mobile Station (MS). The MS computes the 32-bit Signed Response (SRES) based on the encryption of the RAND with the authentication algorithm (A3) using the private subscriber authentication key (Ki). Upon receiving the SRES from the subscriber, the GSM network repeats the calculation to verify the identity of the subscriber. The individual subscriber authentication key (Ki) is never transmitted over the radio channel, as it is present in the subscriber's SIM, as well as the AUC, HLR, and VLR databases. If the received SRES agrees with the calculated value, the MS has been successfully authenticated and may continue. If the values do not match, the connection is terminated and an authentication failure is indicated to the MS. [123]
Signed response calculation is processed within the SIM, which holds confidential subscriber information such as the IMSI or the private subscriber authentication key (Ki). It is never released from the SIM during the authentication process. The SIM contains the ciphering key generating algorithm (A8) that is used to produce the 64-bit ciphering key (Kc). This key is calculated by applying the same random number (RAND) used in the authentication process to ciphering key generating algorithm (A8) with the individual subscriber authentication key (Ki). [123] GSM supports another ciphering algorithm, making the system more resistant to eavesdropping. The ciphering key may be changed at regular intervals if required. As in case of the authentication process, the computation of the ciphering key (Kc) takes place internally within the SIM. Therefore, sensitive information such as the individual subscriber authentication key (Ki) is never revealed by the SIM. [123] Encrypted voice and data communications between the MS and the network is accomplished by using the ciphering algorithm A5. Encrypted communication is initiated by a ciphering mode request command from the GSM network. Upon receipt of this command, the mobile station begins encryption and decryption of data using the ciphering algorithm (A5) and the ciphering key (Kc). [123] To ensure subscriber identity confidentiality, the Temporary Mobile Subscriber Identity (TMSI) is used. Once the authentication and encryption procedures are done, the TMSI is sent to the mobile station. After the receipt, the mobile station responds. The TMSI is valid in the location area in which it was issued. For communications outside the location area, the Location Area Identification (LAI) is necessary in addition to the TMSI. [123]
GSM authentication is shown in Figure 13.

Figure 13: GSM message flow for authentication [10]


HLR
VLR
Authentication Request
Authentication Request
Authentication Request
Response (RAND, SRES, Kc)
Authentication Request (RAND)
Authentication Response (SRES)
Cipher Mode (Turn on Encryption)
Cipher Mode Complete

ProVerif Verification

```
set traceDisplay = long.

query attacker (KC).
query attacker (KI).

query attacker(DecryptedText).
query attacker(DecryptedText2).

query x:bitstring; event(TextDecryption(x)).
query x:bitstring; event(TextDecryption2(x)).

query x:bitstring; event(TextDecryption(x)) ==> event(TextDecryption2(x)).
query x:bitstring; inj-event(TextDecryption(x)) ==> inj-event(
   TextDecryption2(x)).

type key.
type response.
type nonce.

free c:channel.
free AuthReq:bitstring [private].
free KI:key[private].
free KC:key[private].
free AuthResponse:bitstring[private].
free SRES:response[private].
free SRES2:response[private].
free Hello:bitstring[private].
free HelloACK:bitstring[private].
free DecryptedText:bitstring[private].
free DecryptedText2:bitstring[private].

fun GenerateRand(bitstring):bitstring.
fun GenerateCipherKey(key, bitstring): key.
fun Authenticate(key, bitstring):response.
fun Encrypt(bitstring, key):nonce.
reduc forall x:bitstring, y:key; Decrypt(Encrypt(x,y),y)=x.

event RANDGeneration(bitstring).
event CipherKeyGeneration(key).
event ResponseGeneration(response).
event ExpectedResponseGeneration(response).
event ResponseValidation(response).
event TextDecryption(bitstring).
event TextDecryption2(bitstring).

let UE1 =
  out(c, AuthReq).
let Node1 =
  in(c, AuthReq:bitstring);
  out(c, AuthReq).
let VLR1 =
  in(c, AuthReq:bitstring);
  out(c, AuthReq).
let HLR=
  in(c, AuthReq:bitstring);
  new RandomNumber:bitstring;
  let RAND = GenerateRand(RandomNumber) in
    event RANDGeneration(RAND);
  let KC = GenerateCipherKey(KI, RAND) in
    event CipherKeyGeneration(KC);
  let SRES = Authenticate(KI,RAND) in
    event ExpectedResponseGeneration(SRES);
  out(c, (RAND, (KC, SRES))).

let VLR2 =
  in(c, (RAND:bitstring, (KC:key, SRES:response)));
  out(c, (AuthReq, RAND)).

let UE2 =
  in(c, (AuthReq:bitstring, RAND:bitstring));
  let SRES2 = Authenticate(KI, RAND) in
    event ResponseGeneration(SRES2);
  out(c, (AuthResponse, SRES2)).

let VLR3 =
  in(c, (AuthResponse:bitstring, SRES2:response));
  let SRES2 = SRES in
    event ResponseValidation(SRES2);
  if SRES2 = SRES then
  (
    out(c, Encrypt(Hello, KC))
    )
    else
    (
      0
      ).
let UE3 =
  in(c, EncryptedText:nonce);
  let DecryptedText = Decrypt(EncryptedText, KI) in
    event TextDecryption(DecryptedText);
  out(c, Encrypt(HelloACK, KI)).

let VLR4 =
  in(c, EncryptedText2:nonce);
  let DecryptedText2 = Decrypt(EncryptedText2, KC) in
    event TextDecryption2(DecryptedText2).

process (
  !UE1|!Node1|!VLR1|!HLR|!VLR2|!UE2|!VLR3|!UE3|!VLR4
  )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, AuthReq)
) | (
    {3}!
    {4}in(c, AuthReq_16: bitstring);
    {5}out(c, AuthReq_16)
) | (
    {6}!
    {7}in(c, AuthReq_17: bitstring);
    {8}out(c, AuthReq_17)
) | (
    {9}!
    {10}in(c, AuthReq_18: bitstring);
    {11}new RandomNumber: bitstring;
    {12}let RAND: bitstring = GenerateRand(RandomNumber) in
    {13}event RANDGeneration(RAND);
    {14}let KC_19: key = GenerateCipherKey(KI,RAND) in
    {15}event CipherKeyGeneration(KC_19);
    {16}let SRES_20: response = Authenticate(KI,RAND) in
    {17}event ExpectedResponseGeneration(SRES_20);
    {18}out(c, (RAND,(KC_19,SRES_20)))
) | (
    {19}!
    {20}in(c, (RAND_21: bitstring,(KC_22: key,SRES_23: response)));
    {21}out(c, (AuthReq,RAND_21))
) | (
    {22}!
    {23}in(c, (AuthReq_24: bitstring,RAND_25: bitstring));
    {24}let SRES2_26: response = Authenticate(KI,RAND_25) in
    {25}event ResponseGeneration(SRES2_26);
    {26}out(c, (AuthResponse,SRES2_26))
) | (
    {27}!
    {28}in(c, (AuthResponse_27: bitstring,SRES2_28: response));
    {29}let SRES2_29: response = SRES in
    {30}event ResponseValidation(SRES2_29);
    {31}if (SRES2_29 = SRES) then
    {32}out(c, Encrypt(Hello,KC))
) | (
    {33}!
    {34}in(c, EncryptedText: nonce);
    {35}let DecryptedText_30: bitstring = Decrypt(EncryptedText,KI) in
    {36}event TextDecryption(DecryptedText_30);
    {37}out(c, Encrypt(HelloACK,KI))
) | (
    {38}!
    {39}in(c, EncryptedText2: nonce);
    {40}let DecryptedText2_31: bitstring = Decrypt(EncryptedText2,KC) in
    {41}event TextDecryption2(DecryptedText2_31)
)

-- Query not attacker(KC[])
Completing...
Starting query not attacker(KC[])
RESULT not attacker(KC[]) is true.
-- Query not attacker(KI[])
Completing...
Starting query not attacker(KI[])
RESULT not attacker(KI[]) is true.
-- Query not attacker(DecryptedText[])
Completing...
Starting query not attacker(DecryptedText[])
RESULT not attacker(DecryptedText[]) is true.
-- Query not attacker(DecryptedText2[])
Completing...
Starting query not attacker(DecryptedText2[])
RESULT not attacker(DecryptedText2[]) is true.
-- Query not event(TextDecryption(x_32))
Completing...
Starting query not event(TextDecryption(x_32))
RESULT not event(TextDecryption(x_32)) is true.
-- Query not event(TextDecryption2(x_33))
Completing...
Starting query not event(TextDecryption2(x_33))
goal reachable: end(TextDecryption2(Hello[]))

1. The attacker has some term SRES2_1948.
attacker(SRES2_1948).

2. The attacker has some term AuthResponse_1947.
```

```
attacker(AuthResponse_1947).

3. By 2, the attacker may know AuthResponse_1947.
By 1, the attacker may know SRES2_1948.
Using the function 2-tuple the attacker may obtain (AuthResponse_1947,
   SRES2_1948).
attacker((AuthResponse_1947,SRES2_1948)).

4. The message (AuthResponse_1947,SRES2_1948) that the attacker may have
   by 3 may be received at input {28}.
So the message Encrypt(Hello[],KC[]) may be sent to the attacker at output
    {32}.
attacker(Encrypt(Hello[],KC[])).

5. The message Encrypt(Hello[],KC[]) that the attacker may have by 4 may
   be received at input {39}.
So event TextDecryption2(Hello[]) may be executed at {41}.
end(TextDecryption2(Hello[])).


Initial state

Additional knowledge of the attacker:
c
a
a_1953
```

```
New processes:
    (
        !
        out(c, AuthReq)
    ) | (
        !
        in(c, AuthReq_16: bitstring);
        out(c, AuthReq_16)
    ) | (
        !
        in(c, AuthReq_17: bitstring);
        out(c, AuthReq_17)
    ) | (
        !
        in(c, AuthReq_18: bitstring);
        new RandomNumber: bitstring;
        let RAND: bitstring = GenerateRand(RandomNumber) in
        event RANDGeneration(RAND);
        let KC_19: key = GenerateCipherKey(KI,RAND) in
        event CipherKeyGeneration(KC_19);
        let SRES_20: response = Authenticate(KI,RAND) in
        event ExpectedResponseGeneration(SRES_20);
        out(c, (RAND,(KC_19,SRES_20)))
    ) | (
        !
        in(c, (RAND_21: bitstring,(KC_22: key,SRES_23: response)));
        out(c, (AuthReq,RAND_21))
    ) | (
        !
        in(c, (AuthReq_24: bitstring,RAND_25: bitstring));
        let SRES2_26: response = Authenticate(KI,RAND_25) in
        event ResponseGeneration(SRES2_26);
        out(c, (AuthResponse,SRES2_26))
    ) | (
        !
        in(c, (AuthResponse_27: bitstring,SRES2_28: response));
        let SRES2_29: response = SRES in
        event ResponseValidation(SRES2_29);
        if (SRES2_29 = SRES) then
        out(c, Encrypt(Hello,KC))
    ) | (
        !
        in(c, EncryptedText: nonce);
        let DecryptedText_30: bitstring = Decrypt(EncryptedText,KI) in
        event TextDecryption(DecryptedText_30);
        out(c, Encrypt(HelloACK,KI))
    ) | (
        !
        in(c, EncryptedText2: nonce);
        let DecryptedText2_31: bitstring = Decrypt(EncryptedText2,KC) in
        event TextDecryption2(DecryptedText2_31)
    )
```

```
1st process: Reduction |

2nd process: Reduction |
3rd process: Reduction |

4th process: Reduction |

5th process: Reduction |

6th process: Reduction |

7th process: Reduction |

8th process: Reduction |

9th process: Reduction ! 1 copy(ies)

9th process: Beginning of process VLR4

8th process: Reduction ! 0 copy(ies)

7th process: Reduction ! 1 copy(ies)

7th process: Beginning of process VLR3

6th process: Reduction ! 0 copy(ies)

5th process: Reduction ! 0 copy(ies)

4th process: Reduction ! 0 copy(ies)

3rd process: Reduction ! 0 copy(ies)

2nd process: Reduction ! 0 copy(ies)

1st process: Reduction ! 0 copy(ies)

New processes:
(
    in(c, (AuthResponse_1960: bitstring,SRES2_1961: response));
    let SRES2_1962: response = SRES in
    event ResponseValidation(SRES2_1962);
    if (SRES2_1962 = SRES) then
    out(c, Encrypt(Hello,KC))
) | (
    in(c, EncryptedText2_1956: nonce);
    let DecryptedText2_1957: bitstring = Decrypt(EncryptedText2_1956,KC)
       in
    event TextDecryption2(DecryptedText2_1957)
)
```

```
1st process: in(c, (AuthResponse_1960: bitstring,SRES2_1961: response))
   done with message (a,a_1953)

1st process: let SRES2_1974: response = SRES succeeds

1st process: event ResponseValidation(SRES) executed

1st process: if (SRES = SRES) succeeds

1st process: out(c, ~M_1977) with ~M_1977 = Encrypt(Hello,KC) done

Additional knowledge of the attacker:
~M_1977 = Encrypt(Hello,KC)
```

```
1st process: Reduction 0

New processes:
    in(c, EncryptedText2_1956: nonce);
    let DecryptedText2_1957: bitstring = Decrypt(EncryptedText2_1956,KC)
       in
    event TextDecryption2(DecryptedText2_1957)
```

```
1st process: in(c, EncryptedText2_1956: nonce) done with message ~M_1977 =
    Encrypt(Hello,KC)

1st process: let DecryptedText2_1981: bitstring = Hello succeeds

1st process: event TextDecryption2(Hello) executed; it is a goal

New processes:
    0
```

```
The event TextDecryption2(Hello) is executed.
A trace has been found.
RESULT not event(TextDecryption2(x_33)) is false.
-- Query event(TextDecryption(x_34)) ==> event(TextDecryption2(x_34))
Completing...
Starting query event(TextDecryption(x_34)) ==> event(TextDecryption2(x_34)
    )
RESULT event(TextDecryption(x_34)) ==> event(TextDecryption2(x_34)) is
    true.
-- Query inj-event(TextDecryption(x_35)) ==> inj-event(TextDecryption2(
    x_35))
Completing...
Starting query inj-event(TextDecryption(x_35)) ==> inj-event(
    TextDecryption2(x_35))
RESULT inj-event(TextDecryption(x_35)) ==> inj-event(TextDecryption2(x_35)
    ) is true.
```

Appendix 15 – UMTS and LTE Authentication

[43] [149] UMTS and LTE use EPS-AKA [120], which is authentication key agreement protocol designed for UMTS. LTE is reusing the concept of that protocol. EPS-AKA has been shown in Figure 14.
The terms UEA (UMTS Encryption Algorithm) and UIA (UMTS Integrity Algorithm) are used within UMTS as broad categories. UEA1 is a 128-bit block cipher called KASUMI, which is related to the Japanese cipher MISTY. UIA1 is a message authentication code (MAC), also based on KASUMI. UEA2 is a stream cipher related to SNOW 3G, and UIA2 computes a MAC based on the same algorithm. LTE builds upon the lessons learned from deploying the 2G and 3G cryptographic algorithms. [120] [125] [149]
LTE introduced a new set of cryptographic algorithms and a significantly different key structure than that of GSM and UMTS. There are 3 sets of cryptographic algorithms for both confidentiality and integrity termed EPS Encryption Algorithms (EEA) and EPS Integrity Algorithms (EIA). EEA1 and EIA1 are based on SNOW 3G, very similar to algorithms used in UMTS. EEA2 and EIA2 are based on the Advanced Encryption Standard (AES) with EEA2 defined by AES in CTR mode and EIA2 defined by AES-CMAC (Cipherbased MAC). EEA3 and EIA3 are both based on a Chinese cipher ZUC. While these new algorithms have been introduced in LTE, network implementations commonly include older algorithms for backward compatibility for legacy devices and cellular deployments. Many keys in LTE are 256-bits long, but in some current implementations only the 128 least significant bits are used. The specification has allowed for a system-wide upgrade from 128-bit to 256-bit keys. In LTE, the control and user planes may use different algorithms and key sizes. [125]
Given a subscriber secret K (referred as Ki in GSM authentication but essentially the same thing), an authentication sequence number SQN, a random challenge RAND (same as 2G), 3gPP defines a set of algorithms which produce the following authentication and keying materials: [124], [125]

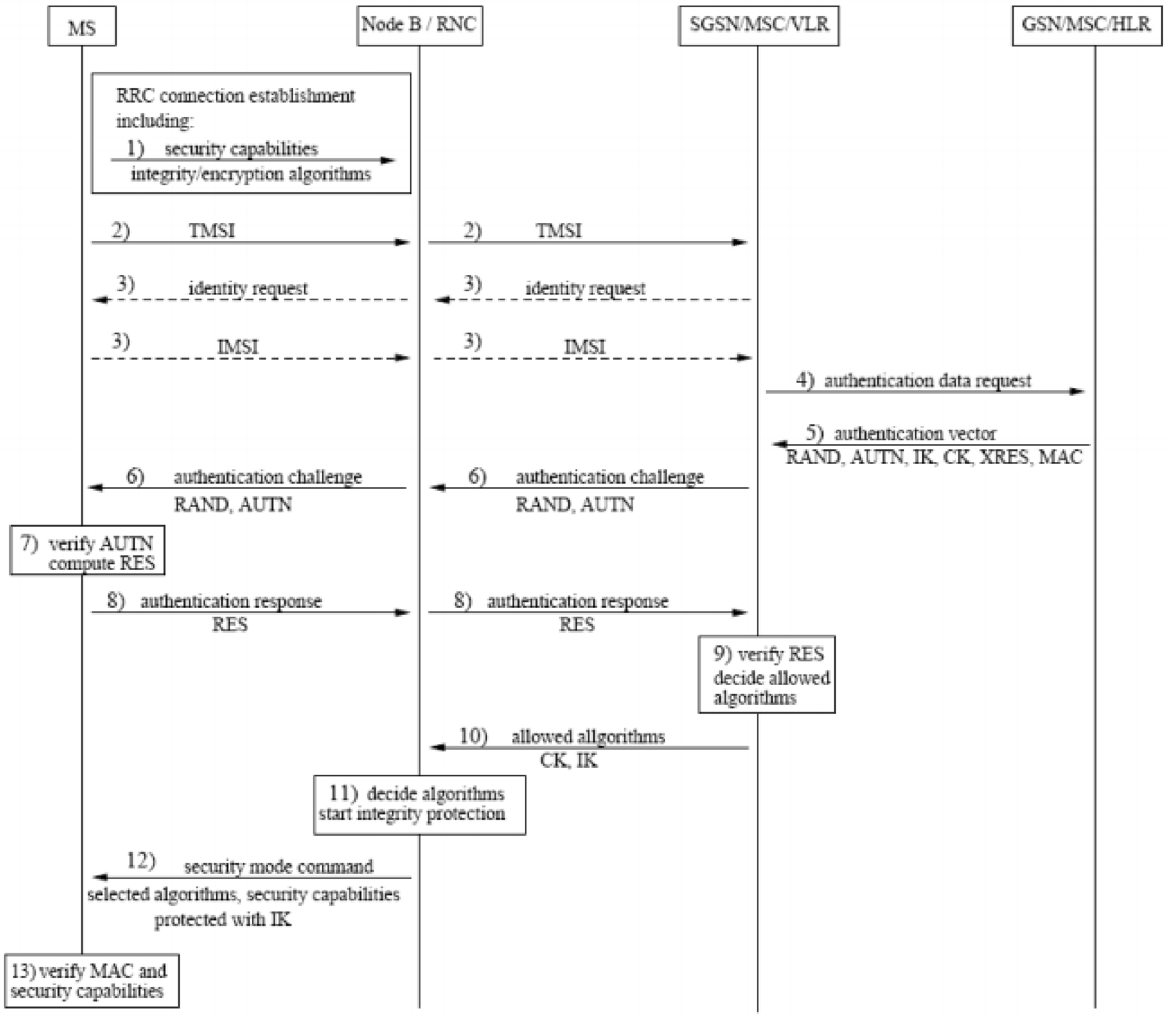
1. MAC: a network authentication code which can be verified by the USIM. Checked by the UE
2. XRES: an expected value returned by the SIM in response of the challenge. Checked by the network.
3. CK: a session key used for ciphering (encrypting) traffic

4. IK: a session key used for marking all traffic packets with a hash signature.
5. AK: an anonymity key used to obfuscate the SQN on its way to the UE

Each one of the above values are generated by a set of five cryptographic functions referred in 3GPP as f1, f2, f3, f4 and f5.
The role of each function is as follows:

1. f1: Generates MAC from K, SQN and RAND. An algorithm variant named AMF can also be used as a salt of the algorithm.
2. f2: Generates XRES from K and RAND.
3. f3: Generates CK from K and RAND.
4. f4: Generates IK from K and RAND.
5. f5: Generates AK from K and RAND.

Figure 14: UMTS and LTE Authentication [10]


MS
Node B / RNC
SGSN/MSC/VLR
GSN/MSC/HLR
RRC connection establishment including:
1) security capabilities integrity/encryption algorithms
2) TMSI
2) TMSI
3) identity request
3) identity request
3) IMSI
3) IMSI
4) authentication data request
5) authentication vector RAND, AUTN, IK, CK, XRES, MAC
6) authentication challenge RAND, AUTN
6) authentication challenge RAND, AUTN
7) verify AUTN compute RES
8) authentication response RES
8) authentication response RES
9) verify RES decide allowed algorithms
10) allowed allgorithms CK, IK
11) decide algorithms start integrity protection
12) security mode command selected algorithms, security capabilities protected with IK
13) verify MAC and security capabilities

ProVerif Verification

```
set traceDisplay = long.

query attacker(PSK).
query attacker(KASME).

query x:bitstring; event(ReceiptDecryption(x)).
query x:bitstring; event(Decryption(x)).

query x:bitstring; event(ReceiptDecryption(x)) ==> event(Decryption(x)).
query x:bitstring; inj-event(ReceiptDecryption(x)) ==> inj-event(
   Decryption(x)).

type key.
type challenge.
type vector.
type response.
type token.

free c:channel.
free ServiceRequest : bitstring[private].
free AuthRequest:bitstring[private].

free IMSI:bitstring [private].
free SNID:bitstring [private].
free AUTN:token [private].
free XRES:response[private].
free RAND:challenge[private].
free AuthResponse:bitstring[private].
free ValidAUTN:token[private].
free Receipt:bitstring[private].
free ReceiptACK:bitstring[private].
free KASME:key[private].
free PSK:key[private].

fun GenerateKASME(bitstring):key.
fun GenerateAuthVector(bitstring, bitstring):vector.
fun GenerateKeySetIdentifier(bitstring):key.
fun ComputeResponse(challenge):bitstring.
fun Anon(challenge):bitstring.
fun GeneratePSK(token, bitstring):key.

fun Encrypt(bitstring, key):response.
reduc forall  x:bitstring, y:key; Decrypt(Encrypt(x,y),y)= x.

event AuthVectorGeneration(vector).
event KASMEGeneration(key).
event eKSIGeneration(key).
event AUTNValidation(token).
event ResponseCalculation(bitstring).
event PSKGeneration(key).
event ReceiptDecryption(bitstring).
event ResponseValidation(response).
event Decryption(bitstring).

let UE1 =
  out(c, ServiceRequest).

let MME1 =
  in(c, ServiceRequest:bitstring);
  out(c, (AuthRequest, (IMSI,SNID))).

let HSS1 =
  in(c, (AuthRequest:bitstring, (IMSI:bitstring, SNID:bitstring)));
  let EPSAV = GenerateAuthVector(AuthRequest, IMSI) in
    event AuthVectorGeneration(EPSAV);
  let KASME = GenerateKASME(SNID) in
    event KASMEGeneration(KASME);
  out(c, (AuthResponse, (EPSAV, (RAND, (XRES, (KASME, AUTN)))))).

let MME2 =
  in(c, (AuthResponse:bitstring, (EPSAV:vector, (RAND:challenge, (XRES:
      response, (KASME:key, AUTN:token))))));
  new randomnumber:bitstring;
  let eKSI = GenerateKeySetIdentifier(randomnumber) in
    event eKSIGeneration(eKSI);
  out(c, (RAND, (AUTN, eKSI))).

let UE2 =
  in(c, (RAND: challenge,(AUTN:token, eKSI:key)));
  let AUTN = ValidAUTN in
    event AUTNValidation(AUTN);
  if AUTN = ValidAUTN then
  (
    let AuthRes = ComputeResponse(RAND) in
      event ResponseCalculation(AuthRes);
    let PSK = GeneratePSK(AUTN, Anon(RAND)) in
      event  PSKGeneration(PSK);
    out(c, Encrypt(Receipt, PSK))
    )
    else
    (
      0
      ).

let MME3 =
  in(c, EncryptedReceipt:response);
  let DecryptedReceipt = Decrypt(EncryptedReceipt, KASME) in
    event ReceiptDecryption(DecryptedReceipt);
  let DecryptedReceipt = XRES in
    event ResponseValidation(DecryptedReceipt);
  if DecryptedReceipt = XRES then
  (
    out(c, Encrypt(ReceiptACK, KASME))
    )
    else
    (
      0
      ).
let UE3 =
  in(c, EncryptedReceiptACK:response);
  let DecryptedReceiptACK = Decrypt(EncryptedReceiptACK, PSK) in
    event Decryption(DecryptedReceiptACK).

process (
  !UE1|!MME1|!HSS1|!MME2|!UE2|!MME3|!UE3
  )
```

ProVerif Result

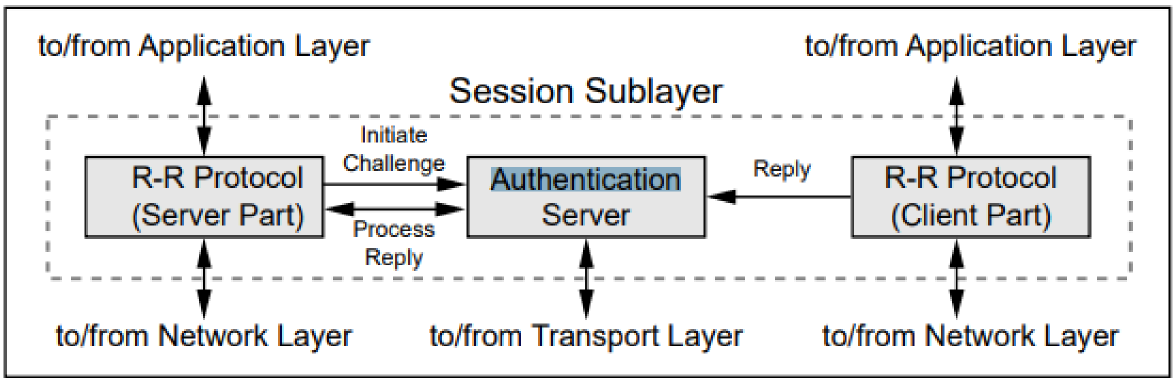
```
Process:
(
    {1}!
    {2}out(c, ServiceRequest)
) | (
    {3}!
    {4}in(c, ServiceRequest_16: bitstring);
    {5}out(c, (AuthRequest,(IMSI,SNID)))
) | (
    {6}!
    {7}in(c, (AuthRequest_17: bitstring,(IMSI_18: bitstring,SNID_19:
       bitstring)));
    {8}let EPSAV: vector = GenerateAuthVector(AuthRequest_17,IMSI_18) in
    {9}event AuthVectorGeneration(EPSAV);
    {10}let KASME_20: key = GenerateKASME(SNID_19) in
    {11}event KASMEGeneration(KASME_20);
    {12}out(c, (AuthResponse,(EPSAV,(RAND,(XRES,(KASME_20,AUTN))))))
) | (
    {13}!
    {14}in(c, (AuthResponse_21: bitstring,(EPSAV_22: vector,(RAND_23:
       challenge,(XRES_24: response,(KASME_25: key,AUTN_26: token))))));
    {15}new randomnumber: bitstring;
    {16}let eKSI: key = GenerateKeySetIdentifier(randomnumber) in
    {17}event eKSIGeneration(eKSI);
    {18}out(c, (RAND_23,(AUTN_26,eKSI)))
) | (
    {19}!
    {20}in(c, (RAND_27: challenge,(AUTN_28: token,eKSI_29: key)));
    {21}let AUTN_30: token = ValidAUTN in
    {22}event AUTNValidation(AUTN_30);
    {23}if (AUTN_30 = ValidAUTN) then
    {24}let AuthRes: bitstring = ComputeResponse(RAND_27) in
    {25}event ResponseCalculation(AuthRes);
    {26}let PSK_31: key = GeneratePSK(AUTN_30,Anon(RAND_27)) in
    {27}event PSKGeneration(PSK_31);
    {28}out(c, Encrypt(Receipt,PSK_31))
) | (
    {29}!
    {30}in(c, EncryptedReceipt: response);
    {31}let DecryptedReceipt: bitstring = Decrypt(EncryptedReceipt,KASME)
```

```
        in
    {32}event ReceiptDecryption(DecryptedReceipt);
    {33}let DecryptedReceipt_32: response = XRES in
    {34}event ResponseValidation(DecryptedReceipt_32);
    {35}if (DecryptedReceipt_32 = XRES) then
    {36}out(c, Encrypt(ReceiptACK,KASME))
) | (
    {37}!
    {38}in(c, EncryptedReceiptACK: response);
    {39}let DecryptedReceiptACK: bitstring = Decrypt(EncryptedReceiptACK,
        PSK) in
    {40}event Decryption(DecryptedReceiptACK)
)

-- Query not attacker(PSK[])
Completing...
Starting query not attacker(PSK[])
RESULT not attacker(PSK[]) is true.
-- Query not attacker(KASME[])
Completing...
Starting query not attacker(KASME[])
RESULT not attacker(KASME[]) is true.
-- Query not event(ReceiptDecryption(x_33))
Completing...
Starting query not event(ReceiptDecryption(x_33))
RESULT not event(ReceiptDecryption(x_33)) is true.
-- Query not event(Decryption(x_34))
Completing...
Starting query not event(Decryption(x_34))
RESULT not event(Decryption(x_34)) is true.
-- Query event(ReceiptDecryption(x_35)) ==> event(Decryption(x_35))
Completing...
Starting query event(ReceiptDecryption(x_35)) ==> event(Decryption(x_35))
RESULT event(ReceiptDecryption(x_35)) ==> event(Decryption(x_35)) is true.
-- Query inj-event(ReceiptDecryption(x_36)) ==> inj-event(Decryption(x_36)
   )
Completing...
Starting query inj-event(ReceiptDecryption(x_36)) ==> inj-event(Decryption
   (x_36))
RESULT inj-event(ReceiptDecryption(x_36)) ==> inj-event(Decryption(x_36))
   is true.
```

Appendix 16 – LonTalk Authentication

When using authenticated messages, the receivers of an authenticated message determine if the sender is authorized to send that message. This can prevent unauthorized access to devices and their applications. This can be used to prevent unauthorized access to devices and their applications. Authentication is implemented by distributing 48-bit keys, one per domain, to the devices at or prior to installation time. For an authenticated message to be accepted by the receiver, both sender and receiver must possess the same key. This key is distinct from the device's Neuron ID. [84] When an authenticated message is sent, the receiver challenges the sender to authenticate itself, using a different random number as a challenge every time. The sender then authenticates by transforming the challenge, using the authentication key along with the data in the original message. The receiver compares the reply to the challenge with its own transformation on the challenge. If the transformations match, the transaction goes forward. This is called an authenticated transaction. The transformation used is designed so that it is extremely difficult to deduce the key, even if the challenge, reply, and authentication algorithm are all known. The use of authentication is configurable individually for each network variable connection. In addition, network management transactions may be optionally authenticated. [84]
With LonTalk, it is up to the sender of the message to initiate an authenticated transaction when required. The sender does this by setting the authentication bit in the message. When a receiver receives a message with the authentication bit set, it must respond with an authentication challenge, even if it does not require the message to be authenticated. It is up to the receiver to determine whether or not the message must be authenticated. This means that a sender may initiate an authenticated transaction on any message, whether required or not. [84] Lontalk authentication sequence is shown in Figure 15:

Figure 15: LonTalk Authentication [10]



ProVerif Verification

```
set traceDisplay = long.

query attacker(AuthenticationKey).

query attacker(NodeBTransformationResult1).
query attacker(NodeATransformationResult1).

query x:nonce; event(NodeBTransformation1(x)).
query x:nonce; event(NodeATransformation1(x)).

query x:nonce; event(NodeBTransformation1(x)) ==> event(
    NodeATransformation1(x)).
query x:nonce; inj-event(NodeBTransformation1(x)) ==> inj-event(
    NodeATransformation1(x)).

type key.
type nonce.

free c:channel.
free AuthenticationKey:key [private].
free AuthenticationRequest:bitstring[private].
free AuthenticationChallenge:bitstring[private].
free NodeBTransformationResult1:nonce[private].
free NodeATransformationResult1:nonce[private].

free Success:bitstring[private].
```

```
event NodeBTransformation1(nonce).
event NodeATransformation1(nonce).

fun Transformation(bitstring, key):nonce.
reduc forall x: bitstring, y:key; DeTransformation(Transformation(x,y), y)
    = x.


let NodeA1 =
  new AuthenticationRequest:bitstring;
  out(c, AuthenticationRequest).

let NodeB1 =
  in(c, AuthenticationRequest:bitstring);
  new AuthenticationChallenge:bitstring;
  let NodeBTransformationResult1 = Transformation(AuthenticationChallenge,
      AuthenticationKey) in
    event NodeBTransformation1(NodeBTransformationResult1);
  out(c, AuthenticationChallenge).

let NodeA2 =
  in(c, AuthenticationChallenge:bitstring);
  let NodeATransformationResult1 = Transformation(AuthenticationChallenge,
      AuthenticationKey) in
    event NodeATransformation1(NodeATransformationResult1);
  out(c, NodeATransformationResult1).

let NodeB2 =
  in(c, NodeATransformationResult1:nonce);
  if NodeATransformationResult1 = NodeBTransformationResult1 then
  (
    out(c,Success)
    )
    else
    (
      0
      ).
process (
  !NodeA1|!NodeB1|!NodeA2|!NodeB2
  )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}new AuthenticationRequest_16: bitstring;
    {3}out(c, AuthenticationRequest_16)
) | (
    {4}!
    {5}in(c, AuthenticationRequest_17: bitstring);
    {6}new AuthenticationChallenge_18: bitstring;
    {7}let NodeBTransformationResult1_19: nonce = Transformation(
       AuthenticationChallenge_18,AuthenticationKey) in
    {8}event NodeBTransformation1(NodeBTransformationResult1_19);
    {9}out(c, AuthenticationChallenge_18)
) | (
    {10}!
    {11}in(c, AuthenticationChallenge_20: bitstring);
    {12}let NodeATransformationResult1_21: nonce = Transformation(
       AuthenticationChallenge_20,AuthenticationKey) in
    {13}event NodeATransformation1(NodeATransformationResult1_21);
    {14}out(c, NodeATransformationResult1_21)
) | (
    {15}!
    {16}in(c, NodeATransformationResult1_22: nonce);
    {17}if (NodeATransformationResult1_22 = NodeBTransformationResult1)
       then
    {18}out(c, Success)
)

-- Query not attacker(AuthenticationKey[])
Completing...
Starting query not attacker(AuthenticationKey[])
RESULT not attacker(AuthenticationKey[]) is true.
-- Query not attacker(NodeBTransformationResult1[])
Completing...
Starting query not attacker(NodeBTransformationResult1[])
RESULT not attacker(NodeBTransformationResult1[]) is true.
-- Query not attacker(NodeATransformationResult1[])
Completing...
Starting query not attacker(NodeATransformationResult1[])
RESULT not attacker(NodeATransformationResult1[]) is true.
```

```
-- Query not event(NodeBTransformation1(x_23))
Completing...
Starting query not event(NodeBTransformation1(x_23))
goal reachable: attacker(AuthenticationRequest_467) -> end(
    NodeBTransformation1(Transformation(AuthenticationChallenge_18[
    AuthenticationRequest_17 = AuthenticationRequest_467,!1 = @sid_468],
    AuthenticationKey[])))
Abbreviations:
AuthenticationChallenge_473 = AuthenticationChallenge_18[
    AuthenticationRequest_17 = AuthenticationRequest_470,!1 = @sid_471]

1. We assume as hypothesis that
attacker(AuthenticationRequest_470).

2. The message AuthenticationRequest_470 that the attacker may have by 1
    may be received at input {5}.
So event NodeBTransformation1(Transformation(AuthenticationChallenge_473,
    AuthenticationKey[])) may be executed at {8}.
end(NodeBTransformation1(Transformation(AuthenticationChallenge_473,
    AuthenticationKey[]))).


Initial state

Additional knowledge of the attacker:
c
a
------------------------------------------------------------------
New processes:
    (
        !
        new AuthenticationRequest_16: bitstring;
        out(c, AuthenticationRequest_16)
    ) | (
        !
        in(c, AuthenticationRequest_17: bitstring);
        new AuthenticationChallenge_18: bitstring;
        let NodeBTransformationResult1_19: nonce = Transformation(
            AuthenticationChallenge_18,AuthenticationKey) in
        event NodeBTransformation1(NodeBTransformationResult1_19);
        out(c, AuthenticationChallenge_18)
    ) | (
        !
        in(c, AuthenticationChallenge_20: bitstring);
        let NodeATransformationResult1_21: nonce = Transformation(
           AuthenticationChallenge_20,AuthenticationKey) in
        event NodeATransformation1(NodeATransformationResult1_21);
        out(c, NodeATransformationResult1_21)
    ) | (
        !
        in(c, NodeATransformationResult1_22: nonce);
        if (NodeATransformationResult1_22 = NodeBTransformationResult1)
           then
        out(c, Success)
    )
```

```
1st process: Reduction |

2nd process: Reduction |

3rd process: Reduction |

4th process: Reduction ! 0 copy(ies)

3rd process: Reduction ! 0 copy(ies)

2nd process: Reduction ! 1 copy(ies)

2nd process: Beginning of process NodeB1

1st process: Reduction ! 0 copy(ies)

New processes:
    in(c, AuthenticationRequest_478: bitstring);
    new AuthenticationChallenge_18: bitstring;
    let NodeBTransformationResult1_479: nonce = Transformation(
       AuthenticationChallenge_18,AuthenticationKey) in
    event NodeBTransformation1(NodeBTransformationResult1_479);
    out(c, AuthenticationChallenge_18)
```

```
1st process: in(c, AuthenticationRequest_478: bitstring) done with message
    a
```

```
1st process: new AuthenticationChallenge_18: bitstring creating
    AuthenticationChallenge_475

1st process: let NodeBTransformationResult1_482: nonce = Transformation(
    AuthenticationChallenge_475,AuthenticationKey) succeeds

1st process: event NodeBTransformation1(Transformation(
    AuthenticationChallenge_475,AuthenticationKey)) executed; it is a goal

New processes:
     out(c, AuthenticationChallenge_475)
```

```
The event NodeBTransformation1(Transformation(AuthenticationChallenge_475,
    AuthenticationKey)) is executed.
A trace has been found.
RESULT not event(NodeBTransformation1(x_23)) is false.
-- Query not event(NodeATransformation1(x_24))
Completing...
Starting query not event(NodeATransformation1(x_24))
goal reachable: attacker(AuthenticationChallenge_594) -> end(
    NodeATransformation1(Transformation(AuthenticationChallenge_594,
    AuthenticationKey[])))

1. We assume as hypothesis that
attacker(AuthenticationChallenge_596).

2. The message AuthenticationChallenge_596 that the attacker may have by 1
     may be received at input {11}.
So event NodeATransformation1(Transformation(AuthenticationChallenge_596,
    AuthenticationKey[])) may be executed at {13}.
end(NodeATransformation1(Transformation(AuthenticationChallenge_596,
    AuthenticationKey[]))).


Initial state

Additional knowledge of the attacker:
c
a_599
```

```
New processes:
    (
        !
        new AuthenticationRequest_16: bitstring;
        out(c, AuthenticationRequest_16)
    ) | (
        !
        in(c, AuthenticationRequest_17: bitstring);
        new AuthenticationChallenge_18: bitstring;
        let NodeBTransformationResult1_19: nonce = Transformation(
           AuthenticationChallenge_18,AuthenticationKey) in
        event NodeBTransformation1(NodeBTransformationResult1_19);
        out(c, AuthenticationChallenge_18)
    ) | (
        !
        in(c, AuthenticationChallenge_20: bitstring);
        let NodeATransformationResult1_21: nonce = Transformation(
           AuthenticationChallenge_20,AuthenticationKey) in
        event NodeATransformation1(NodeATransformationResult1_21);
        out(c, NodeATransformationResult1_21)
    ) | (
        !
        in(c, NodeATransformationResult1_22: nonce);
        if (NodeATransformationResult1_22 = NodeBTransformationResult1)
           then
        out(c, Success)
    )
```

```
1st process: Reduction |

2nd process: Reduction |

3rd process: Reduction |

4th process: Reduction ! 0 copy(ies)

3rd process: Reduction ! 1 copy(ies)

3rd process: Beginning of process NodeA2

2nd process: Reduction ! 0 copy(ies)
```

```
1st process: Reduction ! 0 copy(ies)

New processes:
    in(c, AuthenticationChallenge_603: bitstring);
    let NodeATransformationResult1_604: nonce = Transformation(
       AuthenticationChallenge_603,AuthenticationKey) in
    event NodeATransformation1(NodeATransformationResult1_604);
    out(c, NodeATransformationResult1_604)
```

```
1st process: in(c, AuthenticationChallenge_603: bitstring) done with
   message a_599

1st process: let NodeATransformationResult1_610: nonce = Transformation(
   a_599,AuthenticationKey) succeeds

1st process: event NodeATransformation1(Transformation(a_599,
   AuthenticationKey)) executed; it is a goal

New processes:
    out(c, Transformation(a_599,AuthenticationKey))
```

```
The event NodeATransformation1(Transformation(a_599,AuthenticationKey)) is
    executed.
A trace has been found.
RESULT not event(NodeATransformation1(x_24)) is false.
-- Query event(NodeBTransformation1(x_25)) ==> event(NodeATransformation1(
   x_25))
Completing...
Starting query event(NodeBTransformation1(x_25)) ==> event(
   NodeATransformation1(x_25))
goal reachable: attacker(AuthenticationRequest_723) -> end(
   NodeBTransformation1(Transformation(AuthenticationChallenge_18[
   AuthenticationRequest_17 = AuthenticationRequest_723,!1 = @sid_724],
   AuthenticationKey[])))
Abbreviations:
AuthenticationChallenge_729 = AuthenticationChallenge_18[
   AuthenticationRequest_17 = AuthenticationRequest_726,!1 = @sid_727]

1. We assume as hypothesis that
```

```
attacker(AuthenticationRequest_726).

2. The message AuthenticationRequest_726 that the attacker may have by 1
   may be received at input {5}.
So event NodeBTransformation1(Transformation(AuthenticationChallenge_729,
   AuthenticationKey[])) may be executed at {8}.
end(NodeBTransformation1(Transformation(AuthenticationChallenge_729,
   AuthenticationKey[]))).


Initial state

Additional knowledge of the attacker:
c
a_730
```

```
New processes:
    (
        !
        new AuthenticationRequest_16: bitstring;
        out(c, AuthenticationRequest_16)
    ) | (
        !
        in(c, AuthenticationRequest_17: bitstring);
        new AuthenticationChallenge_18: bitstring;
        let NodeBTransformationResult1_19: nonce = Transformation(
           AuthenticationChallenge_18,AuthenticationKey) in
        event NodeBTransformation1(NodeBTransformationResult1_19);
        out(c, AuthenticationChallenge_18)
    ) | (
        !
        in(c, AuthenticationChallenge_20: bitstring);
        let NodeATransformationResult1_21: nonce = Transformation(
           AuthenticationChallenge_20,AuthenticationKey) in
        event NodeATransformation1(NodeATransformationResult1_21);
        out(c, NodeATransformationResult1_21)
    ) | (
        !
        in(c, NodeATransformationResult1_22: nonce);
        if (NodeATransformationResult1_22 = NodeBTransformationResult1)
           then
        out(c, Success)
    )
```

```
1st process: Reduction |

2nd process: Reduction |

3rd process: Reduction |

4th process: Reduction ! 0 copy(ies)

3rd process: Reduction ! 0 copy(ies)

2nd process: Reduction ! 1 copy(ies)

2nd process: Beginning of process NodeB1

1st process: Reduction ! 0 copy(ies)

New processes:
    in(c, AuthenticationRequest_736: bitstring);
    new AuthenticationChallenge_18: bitstring;
    let NodeBTransformationResult1_737: nonce = Transformation(
       AuthenticationChallenge_18,AuthenticationKey) in
    event NodeBTransformation1(NodeBTransformationResult1_737);
    out(c, AuthenticationChallenge_18)
```

```
1st process: in(c, AuthenticationRequest_736: bitstring) done with message
     a_730

1st process: new AuthenticationChallenge_18: bitstring creating
    AuthenticationChallenge_732

1st process: let NodeBTransformationResult1_742: nonce = Transformation(
    AuthenticationChallenge_732,AuthenticationKey) succeeds

1st process: event NodeBTransformation1(Transformation(
    AuthenticationChallenge_732,AuthenticationKey)) executed; it is a goal

New processes:
    out(c, AuthenticationChallenge_732)
```

```
The event NodeBTransformation1(Transformation(AuthenticationChallenge_732,
   AuthenticationKey)) is executed.
A trace has been found.
RESULT event(NodeBTransformation1(x_25)) ==> event(NodeATransformation1(
   x_25)) is false.
-- Query inj-event(NodeBTransformation1(x_26)) ==> inj-event(
   NodeATransformation1(x_26))
Completing...
Starting query inj-event(NodeBTransformation1(x_26)) ==> inj-event(
   NodeATransformation1(x_26))
goal reachable: attacker(AuthenticationRequest_858) -> end(endsid_859,
   NodeBTransformation1(Transformation(AuthenticationChallenge_18[
   AuthenticationRequest_17 = AuthenticationRequest_858,!1 = endsid_859],
   AuthenticationKey[])))
Abbreviations:
AuthenticationChallenge_866 = AuthenticationChallenge_18[
   AuthenticationRequest_17 = AuthenticationRequest_862,!1 = endsid_864]

1. We assume as hypothesis that
attacker(AuthenticationRequest_862).

2. The message AuthenticationRequest_862 that the attacker may have by 1
   may be received at input {5}.
So event NodeBTransformation1(Transformation(AuthenticationChallenge_866,
   AuthenticationKey[])) may be executed at {8} in session endsid_864.
end(endsid_864,NodeBTransformation1(Transformation(
   AuthenticationChallenge_866,AuthenticationKey[]))).


Initial state

Additional knowledge of the attacker:
c
a_868
```

```
New processes:
    (
        !
        new AuthenticationRequest_16: bitstring;
        out(c, AuthenticationRequest_16)
    ) | (
        !
        in(c, AuthenticationRequest_17: bitstring);
        new AuthenticationChallenge_18: bitstring;
        let NodeBTransformationResult1_19: nonce = Transformation(
           AuthenticationChallenge_18,AuthenticationKey) in
        event NodeBTransformation1(NodeBTransformationResult1_19);
        out(c, AuthenticationChallenge_18)
    ) | (
        !
        in(c, AuthenticationChallenge_20: bitstring);
        let NodeATransformationResult1_21: nonce = Transformation(
           AuthenticationChallenge_20,AuthenticationKey) in
        event NodeATransformation1(NodeATransformationResult1_21);
        out(c, NodeATransformationResult1_21)
    ) | (
        !
        in(c, NodeATransformationResult1_22: nonce);
        if (NodeATransformationResult1_22 = NodeBTransformationResult1)
           then
        out(c, Success)
    )
```

```
1st process: Reduction |

2nd process: Reduction |

3rd process: Reduction |

4th process: Reduction ! 0 copy(ies)

3rd process: Reduction ! 0 copy(ies)

2nd process: Reduction ! 1 copy(ies)

2nd process: Beginning of process NodeB1

1st process: Reduction ! 0 copy(ies)

New processes:
    in(c, AuthenticationRequest_873: bitstring);
    new AuthenticationChallenge_18: bitstring;
    let NodeBTransformationResult1_874: nonce = Transformation(
       AuthenticationChallenge_18,AuthenticationKey) in
    event NodeBTransformation1(NodeBTransformationResult1_874);
    out(c, AuthenticationChallenge_18)
```

```
1st process: in(c, AuthenticationRequest_873: bitstring) done with message
    a_868

1st process: new AuthenticationChallenge_18: bitstring creating
   AuthenticationChallenge_869

1st process: let NodeBTransformationResult1_879: nonce = Transformation(
   AuthenticationChallenge_869,AuthenticationKey) succeeds

1st process: event NodeBTransformation1(Transformation(
   AuthenticationChallenge_869,AuthenticationKey)) executed; it is a goal

New processes:
    out(c, AuthenticationChallenge_869)
```

```
The event NodeBTransformation1(Transformation(AuthenticationChallenge_869,
   AuthenticationKey)) is executed in session a_867.
A trace has been found.
RESULT inj-event(NodeBTransformation1(x_26)) ==> inj-event(
   NodeATransformation1(x_26)) is false.
RESULT (even event(NodeBTransformation1(x_860)) ==> event(
   NodeATransformation1(x_860)) is false.)
```

Appendix 17 – UWB Authentication

ECMA-368 standard specifies a mutual authentication mechanism. Mutual authentication is completed through 4-way handshake process. If both parts share the same master key, authentication will be successful. This is a message authentication scheme and can be applied to distributed systems. Message integrity code (MIC) also serves as authenticator. It is calculated by Advanced Encryption Standard(AES) with cipher block chaining(CBC) mode, and encrypted by counter mode. This scheme will be a long-term security solution recommended in IEEE 802.11i. Confidentiality deals with the attacks of disclosure and traffic analysis through encryption. For unicast traffic, this key is pair-wise temporal key(PTK), and group temporal key(GTK) in broadcast or multicast traffic. PTK or GTK is not same as the master key, though they are derived from the master key. A master key is mapped to a master key identifier(MKID). A device can select a encryption offset(EO) part in a frame not to be encrypted. Applying integrity to selected fields of a message, a message integrity code(MIC) is produced. The MIC, also known as message authentication code(MAC), is an 8-octet cryptographic checksum and is used to protect the integrity of the MAC Header and frame Payload. The symmetric encrypted MIC provides not only authentication but also confidentiality.
CCM authentication is described in Appendix 19.

ProVerif Verification

```
set traceDisplay=long.

query attacker(PTKResponder).
query attacker(PTKInitiator).
query attacker(DecryptedData1).
query attacker(DecryptedData2).

query x:bitstring; event(DataDecryption1(x)).
query x:bitstring; event(DataDecryption2(x)).
query x:bitstring; event(DataDecryption1(x)) ==> event(DataDecryption2(x))
    .
query x:bitstring; inj-event(DataDecryption1(x)) ==> inj-event(
    DataDecryption2(x)).

type nonce.
type key.
type MIC.

free TKID:bitstring [private].
free UniqueTKID:bitstring [private].
free PTKMic:MIC[private].
free PTKMic2:MIC[private].
free PTKMic3:MIC[private].
free PTKMic4, PTKMic5:MIC[private].
free Success:bitstring[private].

free PTKResponder:key [private].
free PTKInitiator:key [private].
free INonce, INonce2, INonce3:nonce [private].
free RNonce, RNonce2, RNonce3:nonce [private].
free MKID:bitstring[private].
free StatusCode:bitstring[private].
free DecryptedData1:bitstring[private].
free DecryptedData2:bitstring[private].
free TEST:bitstring[private].
event VerifyMKIDUniqueness(bitstring).
event PTKMicVerification(MIC).
event PTKMicVerification2(MIC).
event PTKMicVerification3(MIC).
event PTKMicVerification4(MIC).
event PTKMicVerification5(MIC).
event DataDecryption1(bitstring).
event DataDecryption2(bitstring).

free c:channel.

fun AESEnc(bitstring, nonce, key): nonce.
reduc forall x: bitstring, y:nonce, z:key; AESDec(AESEnc(x,y,z),z) = x.

let Initiator =
  out(c, (MKID, (TKID,INonce))).

let Responder =
  in(c, (MKID:bitstring, (TKID:bitstring,INonce:nonce)));
  let TKID = UniqueTKID in
    event VerifyMKIDUniqueness(TKID);
  if TKID = UniqueTKID then
  (
    out(c, (StatusCode, RNonce))
    )
    else
    (
      0
      ).

let Initiator2 =
  in(c, (StatusCode:bitstring, RNonce:nonce));
  let PTKMic = PTKMic in
    event PTKMicVerification(PTKMic);
  if PTKMic = PTKMic then
  (
    out(c, INonce2)
    )
    else
    (
      0
      ).

  let Responder2 =
    in(c, INonce2:nonce);
    let PTKMic2 = PTKMic2 in
      event PTKMicVerification2(PTKMic2);
    if PTKMic2 = PTKMic2 then
    (
      out(c, RNonce2)
      )
      else
      (
        0
        ).

let Initiator3 =
    in(c, RNonce2:nonce);
    let PTKMic3 = PTKMic3 in
      event PTKMicVerification3(PTKMic3);
    if PTKMic3 = PTKMic3 then
    (
      out(c, AESEnc(TEST, INonce3, PTKInitiator))
      )
      else
      (
        0
        ).

    let Responder3 =
      in(c, EncryptedData1:nonce);
      let PTKMic4 = PTKMic4 in
        event PTKMicVerification4(PTKMic4);
      if PTKMic4 = PTKMic4 then
      (
        let DecryptedData1 = AESDec(EncryptedData1, PTKResponder) in
          event DataDecryption1(DecryptedData1);
        out(c, AESEnc(Success, RNonce3, PTKResponder))
        )
        else
        (
          0
          ).

let Initiator4 =
    in(c, EncryptedData2:nonce);
    let PTKMic5 = PTKMic5 in
      event PTKMicVerification5(PTKMic5);
    if PTKMic5 = PTKMic5 then
    (
      let DecryptedData2 = AESDec(EncryptedData2, PTKInitiator) in
        event DataDecryption2(DecryptedData2)
      )
      else
      (
        0
        ).

process
  (
    !Initiator|!Responder|!Initiator2|!Responder2|!Initiator3|!Responder3
       |!Initiator4
    )
```

ProVerif Result

```
Process:
(
    {1}!
    {2}out(c, (MKID,(TKID,INonce)))
) | (
    {3}!
    {4}in(c, (MKID_16: bitstring,(TKID_17: bitstring,INonce_18: nonce)));
    {5}let TKID_19: bitstring = UniqueTKID in
    {6}event VerifyMKIDUniqueness(TKID_19);
    {7}if (TKID_19 = UniqueTKID) then
    {8}out(c, (StatusCode,RNonce))
) | (
    {9}!
    {10}in(c, (StatusCode_20: bitstring,RNonce_21: nonce));
    {11}let PTKMic_22: MIC = PTKMic in
    {12}event PTKMicVerification(PTKMic_22);
    {13}if (PTKMic_22 = PTKMic_22) then
    {14}out(c, INonce2)
) | (
    {15}!
    {16}in(c, INonce2_23: nonce);
    {17}let PTKMic2_24: MIC = PTKMic2 in
    {18}event PTKMicVerification2(PTKMic2_24);
    {19}if (PTKMic2_24 = PTKMic2_24) then
    {20}out(c, RNonce2)
) | (
    {21}!
    {22}in(c, RNonce2_25: nonce);
    {23}let PTKMic3_26: MIC = PTKMic3 in
    {24}event PTKMicVerification3(PTKMic3_26);
    {25}if (PTKMic3_26 = PTKMic3_26) then
    {26}out(c, AESEnc(TEST,INonce3,PTKInitiator))
) | (
    {27}!
    {28}in(c, EncryptedData1: nonce);
    {29}let PTKMic4_27: MIC = PTKMic4 in
    {30}event PTKMicVerification4(PTKMic4_27);
    {31}if (PTKMic4_27 = PTKMic4_27) then
    {32}let DecryptedData1_28: bitstring = AESDec(EncryptedData1,
       PTKResponder) in
    {33}event DataDecryption1(DecryptedData1_28);
    {34}out(c, AESEnc(Success,RNonce3,PTKResponder))
) | (
    {35}!
    {36}in(c, EncryptedData2: nonce);
    {37}let PTKMic5_29: MIC = PTKMic5 in
    {38}event PTKMicVerification5(PTKMic5_29);
    {39}if (PTKMic5_29 = PTKMic5_29) then
    {40}let DecryptedData2_30: bitstring = AESDec(EncryptedData2,
        PTKInitiator) in
    {41}event DataDecryption2(DecryptedData2_30)
)

-- Query not attacker(PTKResponder[])
Completing...
Starting query not attacker(PTKResponder[])
RESULT not attacker(PTKResponder[]) is true.
-- Query not attacker(PTKInitiator[])
Completing...
Starting query not attacker(PTKInitiator[])
RESULT not attacker(PTKInitiator[]) is true.
-- Query not attacker(DecryptedData1[])
Completing...
Starting query not attacker(DecryptedData1[])
RESULT not attacker(DecryptedData1[]) is true.
-- Query not attacker(DecryptedData2[])
Completing...
Starting query not attacker(DecryptedData2[])
RESULT not attacker(DecryptedData2[]) is true.
-- Query not event(DataDecryption1(x_31))
Completing...
Starting query not event(DataDecryption1(x_31))
RESULT not event(DataDecryption1(x_31)) is true.
-- Query not event(DataDecryption2(x_32))
Completing...
Starting query not event(DataDecryption2(x_32))
goal reachable: end(DataDecryption2(TEST[]))

1. The attacker has some term RNonce2_1755.
attacker(RNonce2_1755).

2. The message RNonce2_1755 that the attacker may have by 1 may be
```

```
    received at input {22}.
So the message AESEnc(TEST[],INonce3[],PTKInitiator[]) may be sent to the
    attacker at output {26}.
attacker(AESEnc(TEST[],INonce3[],PTKInitiator[])).

3. The message AESEnc(TEST[],INonce3[],PTKInitiator[]) that the attacker
    may have by 2 may be received at input {36}.
So event DataDecryption2(TEST[]) may be executed at {41}.
end(DataDecryption2(TEST[])).


Initial state

Additional knowledge of the attacker:
c
a
```

```
New processes:
    (
        !
        out(c, (MKID,(TKID,INonce)))
    ) | (
        !
        in(c, (MKID_16: bitstring,(TKID_17: bitstring,INonce_18: nonce)));
        let TKID_19: bitstring = UniqueTKID in
        event VerifyMKIDUniqueness(TKID_19);
        if (TKID_19 = UniqueTKID) then
        out(c, (StatusCode,RNonce))
    ) | (
        !
        in(c, (StatusCode_20: bitstring,RNonce_21: nonce));
        let PTKMic_22: MIC = PTKMic in
        event PTKMicVerification(PTKMic_22);
        if (PTKMic_22 = PTKMic_22) then
        out(c, INonce2)
    ) | (
        !
        in(c, INonce2_23: nonce);
        let PTKMic2_24: MIC = PTKMic2 in
        event PTKMicVerification2(PTKMic2_24);
        if (PTKMic2_24 = PTKMic2_24) then
        out(c, RNonce2)
    ) | (
        !
        in(c, RNonce2_25: nonce);
        let PTKMic3_26: MIC = PTKMic3 in
        event PTKMicVerification3(PTKMic3_26);
        if (PTKMic3_26 = PTKMic3_26) then
        out(c, AESEnc(TEST,INonce3,PTKInitiator))
    ) | (
        !
        in(c, EncryptedData1: nonce);
        let PTKMic4_27: MIC = PTKMic4 in
        event PTKMicVerification4(PTKMic4_27);
        if (PTKMic4_27 = PTKMic4_27) then
        let DecryptedData1_28: bitstring = AESDec(EncryptedData1,
            PTKResponder) in
        event DataDecryption1(DecryptedData1_28);
        out(c, AESEnc(Success,RNonce3,PTKResponder))
    ) | (
        !
        in(c, EncryptedData2: nonce);
        let PTKMic5_29: MIC = PTKMic5 in
        event PTKMicVerification5(PTKMic5_29);
        if (PTKMic5_29 = PTKMic5_29) then
        let DecryptedData2_30: bitstring = AESDec(EncryptedData2,
            PTKInitiator) in
        event DataDecryption2(DecryptedData2_30)
    )
```

```
1st process: Reduction |

2nd process: Reduction |

3rd process: Reduction |

4th process: Reduction |

5th process: Reduction |

6th process: Reduction |

7th process: Reduction ! 1 copy(ies)
```

```
7th process: Beginning of process Initiator4

6th process: Reduction ! 0 copy(ies)

5th process: Reduction ! 1 copy(ies)

5th process: Beginning of process Initiator3

4th process: Reduction ! 0 copy(ies)

3rd process: Reduction ! 0 copy(ies)

2nd process: Reduction ! 0 copy(ies)

1st process: Reduction ! 0 copy(ies)

New processes:
(
    in(c, RNonce2_1768: nonce);
    let PTKMic3_1769: MIC = PTKMic3 in
    event PTKMicVerification3(PTKMic3_1769);
    if (PTKMic3_1769 = PTKMic3_1769) then
    out(c, AESEnc(TEST,INonce3,PTKInitiator))
) | (
    in(c, EncryptedData2_1763: nonce);
    let PTKMic5_1764: MIC = PTKMic5 in
    event PTKMicVerification5(PTKMic5_1764);
    if (PTKMic5_1764 = PTKMic5_1764) then
    let DecryptedData2_1765: bitstring = AESDec(EncryptedData2_1763,
       PTKInitiator) in
    event DataDecryption2(DecryptedData2_1765)
)
```

```
1st process: in(c, RNonce2_1768: nonce) done with message a

1st process: let PTKMic3_1775: MIC = PTKMic3 succeeds

1st process: event PTKMicVerification3(PTKMic3) executed

1st process: if (PTKMic3 = PTKMic3) succeeds
```

```
1st process: out(c, ~M_1778) with ~M_1778 = AESEnc(TEST,INonce3,
   PTKInitiator) done

Additional knowledge of the attacker:
~M_1778 = AESEnc(TEST,INonce3,PTKInitiator)
```

```
1st process: Reduction 0

New processes:
    in(c, EncryptedData2_1763: nonce);
    let PTKMic5_1764: MIC = PTKMic5 in
    event PTKMicVerification5(PTKMic5_1764);
    if (PTKMic5_1764 = PTKMic5_1764) then
    let DecryptedData2_1765: bitstring = AESDec(EncryptedData2_1763,
       PTKInitiator) in
    event DataDecryption2(DecryptedData2_1765)
```

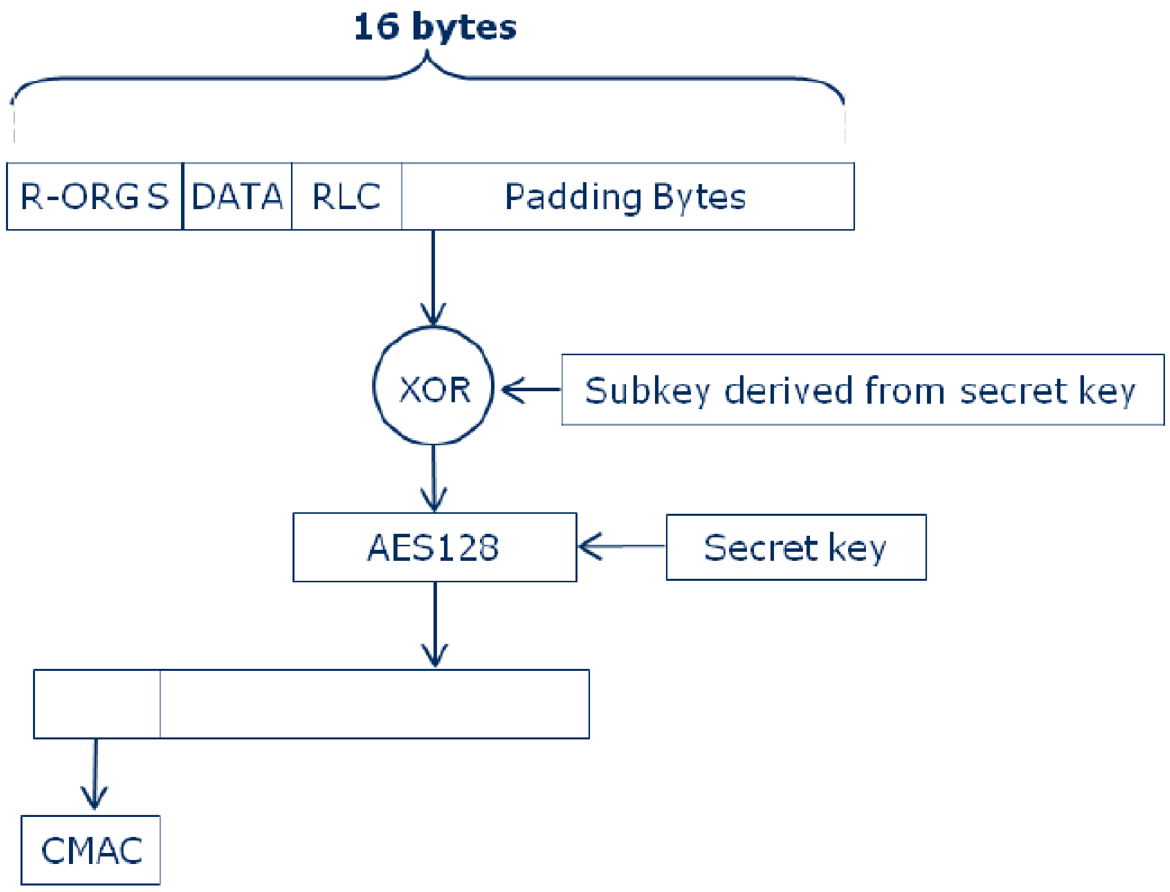
```
1st process: in(c, EncryptedData2_1763: nonce) done with message ~M_1778 =
    AESEnc(TEST,INonce3,PTKInitiator)

1st process: let PTKMic5_1782: MIC = PTKMic5 succeeds

1st process: event PTKMicVerification5(PTKMic5) executed

1st process: if (PTKMic5 = PTKMic5) succeeds

1st process: let DecryptedData2_1784: bitstring = TEST succeeds

1st process: event DataDecryption2(TEST) executed; it is a goal

New processes:
    0
```

```
The event DataDecryption2(TEST) is executed.
A trace has been found.
RESULT not event(DataDecryption2(x_32)) is false.
-- Query event(DataDecryption1(x_33)) ==> event(DataDecryption2(x_33))
Completing...
Starting query event(DataDecryption1(x_33)) ==> event(DataDecryption2(x_33
   ))
RESULT event(DataDecryption1(x_33)) ==> event(DataDecryption2(x_33)) is
   true.
-- Query inj-event(DataDecryption1(x_34)) ==> inj-event(DataDecryption2(
   x_34))
Completing...
Starting query inj-event(DataDecryption1(x_34)) ==> inj-event(
   DataDecryption2(x_34))
RESULT inj-event(DataDecryption1(x_34)) ==> inj-event(DataDecryption2(x_34
   )) is true.
```

Appendix 18 – CMAC Authentication

CMAC is a message authentication code (MAC), which is constructed with a block cipher using its own cipher algorithm.
EnOcean has described CMAC authentication as follows, which has been shown in Figure 16: MAC generation algorithm for a message ¡= 16 bytes.
The R-ORG-S is the secure radio message R-ORG code.
DATA is the secure message DATA field.
XOR is used to encrypt message with a different key.
If the receiving device has a shared secret key, it is able to decrypt the message.

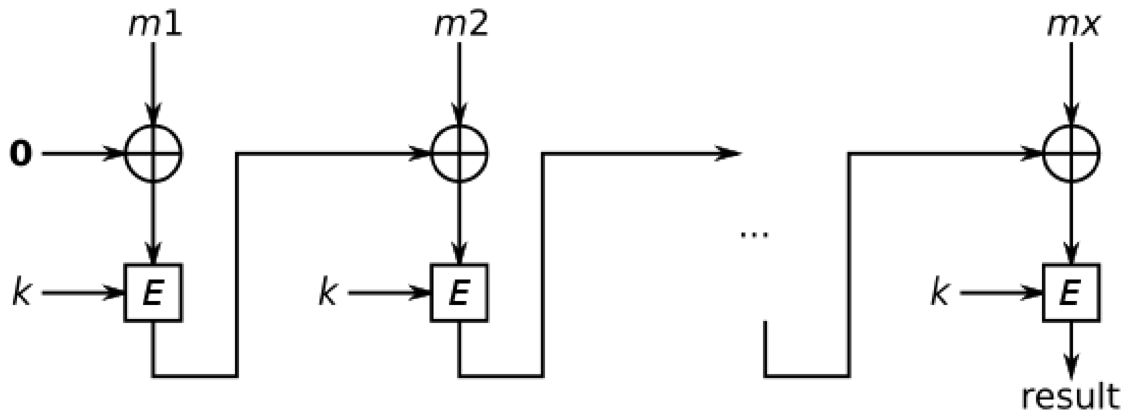
Figure 16: CMAC message calculation [4]



Appendix 19 – CCM Authentication

CBC-MAC is a method for constructing a message authentication code from a block cipher. With CBC (Cipher Block Chaining)-MAC (Message Authentication Code) messages are authenticated with a secret shared key. Messages are encrypted with the standard form of AES and then throw away everything apart from the last block, and use this as a fixed-length MAC. If the key is not secret the method provides little in the way of security.
Algorithm calculation has been shown in Figure 17.

Figure 17: CBC-MAC message calculation [11]



Appendix 20 – BAN logic Verification for Diffie-Hellman Key Exchange

Initial Assumptions:

$A \mid\equiv \xmapsto{K_A} A$

$A \mid\equiv \xmapsto{K_B} B$

$B \mid\equiv \xmapsto{K_B} B$

$B \mid\equiv \xmapsto{K_A} A$

$A \mid\equiv B \Rightarrow G^{N_B}$

$B \mid\equiv A \Rightarrow G^{N_A}$

Idealized Protocol:

Message 1: $A \triangleleft G^{N_B}$

Message 2: $B \triangleleft G^{N_A}, \{G^{N_A}, G^{N_B}\}_{K_{A^{-1}}}$

Message 3:$A \triangleleft G^{N_B}, \{G^{N_B}, G^{N_A}\}_{K_{B^{-1}}}$

Protocol Goal:

$A \mid\equiv A \xleftrightarrow{K_{AB}} B$

$B \mid\equiv A \xleftrightarrow{K_{AB}} B$

$B \mid\equiv A \mid\equiv A \xleftrightarrow{K_{AB}} B$

$A \mid\equiv B \mid\equiv A \xleftrightarrow{K_{AB}} B$

Verification:

In first message of idealized protocol, B chooses random N_B and calculates G^{N_B} and then sends it to A, therefor

$B \mid\equiv N_B$

$B \mid\equiv \#(N_B)$

$A \triangleleft G^{N_B}$

In second message of idealized protocol, A chooses randomN_A and calculates G^{N_A}, then sends and believes

$A \mid\equiv N_A$

$A \mid\equiv \#(N_A)$

$B \triangleleft G^{N_A}, \{G^{N_A}, G^{N_B}\}_{K_{A^{-1}}}$

From the last sent message by A to B, we can obtain by applying message-meaning rules and initial assumptions $B \mid\equiv \xmapsto{K_A} A$ that

$B \mid\equiv A \mid\sim (G^{N_A}, G^{N_B})$

Applying freshness conjucation to the $B \mid\equiv \#(N_B)$, then it is possible to obtain:

$B \mid\equiv \#(G^{N_B}, G^{N_A})$

Applying nonce verification rule to $B \mid\equiv A \mid\sim (G^{N_A}, G^{N_B})$ and $B \mid\equiv \#(G^{N_B,G^{N_A}})$, then it can be obtained that

$B \mid\equiv A \mid\equiv (G^{N_A}, G^{N_B})$

From decomposition, it can be obtained that

$B \mid\equiv A \mid\equiv G^{N_A}$

$B \mid\equiv A \mid\equiv N_A$

From $B \mid\equiv A \mid\equiv G^{N_A}$ and initial assumption of $B \mid\equiv A \Rightarrow G^{N_A}$ and applying jurisdiction rule, it is obtained that

$B \mid\equiv G^{N_A}$

From $B \mid\equiv \#(G^{N_B,G^{N_A}})$ it can be obtained that

$B \mid\equiv A \mid\equiv G^{N_B}$

From third message of idealized protocol it is possible to derive that

$A \triangleleft G^{N_B}, \{G^{N_A}, G^{N_B}\}_{K_{B^{-1}}}$

From initial assumption of $A \mid\equiv \xmapsto{K_B} B$ and applying message-meaning rule, it can be obtained that

$A \mid\equiv B \mid\sim (G^{N_A}, G^{N_B})$

Applying freshness conjucation to $A \mid\equiv \#(N_A)$, it is obtained that

$A \mid\equiv \#(G^{N_A}, G^{N_B})$

From nonce verification of $B \mid\equiv \#(G^{N_B}, G^{N_A})$ and $A \mid\equiv B \mid\sim (G^{N_A}, G^{N_B})$ it is obtained that

$A \mid\equiv B \mid\equiv (G^{N_B}, G^{N_A})$

From decomposition of last message, it is obtained that

$A \mid\equiv B \mid\equiv G^{N_B}$

$A \mid\equiv B \mid\equiv N_B$

From $A \mid\equiv B \mid\equiv G^{N_B}$, initial assumptions of $A \mid\equiv B \Rightarrow G^{N_B}$ and by applying jurisdiction rule :

$A \mid\equiv G^{N_B}$

From decomposition of $A \mid\equiv B \mid\equiv (G^{N_B}, G^{N_A})$ it is obtained that :

$A \mid\equiv B \mid\equiv G^{N_A}$

A calculates now $K_{AB} = (G^{N_B})^{N_A}$.

From messages $A \triangleleft G^{N_B}$ and $A \mid\equiv \#(N_A)$ with applying freshness conjucation, it is obtained that

$A \mid\equiv \#(K_{AB})$

From $A \mid\equiv \#(K_{AB})$ and $A \mid\equiv B \mid\equiv N_B$ by applying shared key rule :

$A \mid\equiv A \xleftrightarrow{K_{AB}} B$

Due to symmetrical key exchange protocol B is bound to does the same.
$B \mid\equiv A \mid\equiv A \xleftrightarrow{K_{AB}} B$
B calculates now $K_{AB} = (G^{N_A})^{N_B}$.
$B \mid\equiv \#(K_{AB})$
From $B \mid\equiv \#(K_{AB})$ and $B \mid\equiv \#(N_B)$ by applying shared key rule :
$B \mid\equiv A \xleftrightarrow{K_{AB}} B$
Due to symmetrical key exchange protocol A is bound to does the same.
$A \mid\equiv B \mid\equiv A \xleftrightarrow{K_{AB}} B$
With these deductions following goals are reached :
$A \mid\equiv A \xleftrightarrow{K_{AB}} B$
$B \mid\equiv A \xleftrightarrow{K_{AB}} B$
$B \mid\equiv A \mid\equiv A \xleftrightarrow{K_{AB}} B$
$A \mid\equiv B \mid\equiv A \xleftrightarrow{K_{AB}} B$
Which means, that goals of the protocol are reached.

Appendix 21 – BAN logic Verification WiMAX PKMv1 Key Exchange

Initial Assumptions:

$SS \mid\equiv \overset{K_A}{\longmapsto} A$

$BS \mid\equiv \overset{K_A}{\longmapsto} A$

$SS \mid\equiv \overset{K_{SS}}{\longmapsto} SS$

$BS \mid\equiv \overset{K_{SS}}{\longmapsto} SS$

$SS \mid\equiv A \Rightarrow \overset{K}{\longmapsto} BS$

$BS \mid\equiv A \Rightarrow \overset{K}{\longmapsto} SS$

$SS \mid\equiv BS \Rightarrow (SS \overset{K}{\leftrightarrow} BS)$

$BS \mid\equiv SS \overset{K}{\leftrightarrow} BS$

$SS \mid\equiv \#(T_{ABS})$

$SS \mid\equiv \#(N_{SS})$

$SS \mid\equiv \#(T_{BS})$

$BS \mid\equiv \#(T_{A_{SS}})$

$BS \mid\equiv \#(N_{BS})$

$BS \mid\equiv \#(T_{SS})$

Idealized Protocol:

Message 1: $BS \triangleleft SS.REQ, \{T_{A_{SS}}, \overset{K_{SS}}{\longmapsto} SS\}_{K_{A^{-1}}}$

Message 2: $SS \triangleleft \{SS \overset{AK}{\longleftrightarrow} BS\}_{K_{SS}}$

Protocol Goal:

$SS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

$BS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

$BS \mid\equiv SS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

$SS \mid\equiv BS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

Verification:

$BS \triangleleft SS \ \rightarrow \ BS \ : \ SS.REQ, \{T_{A_{SS}}, \overset{K_{SS}}{\longmapsto} SS\}_{K_{A^{-1}}}$ and by initial assumption of $BS \mid\equiv \overset{K_A}{\longmapsto} A$ it can be deduced with message-meaning rule that:

$BS \mid\equiv A \mid\sim (T_{A_{SS}}, \overset{K_{SS}}{\longmapsto} SS)$

With initial assumption of $BS \mid\equiv \#(T_{A_{SS}})$ and last message, while applying nonce verification rule, it can be deduced that

$BS \mid\equiv A \mid\equiv \overset{K_{SS}SS}{\longmapsto}$

Applying rule of jurisdiction and initial assumption to last deduction :

$SS \triangleleft SS \overset{AK}{\longleftrightarrow} BS$

No more assertions can be added and the result of this verification is that SS sees that there is a key AK, which can be used to communicate with BS, but does not know if it is assigned by BS or not, because the identity has not been included in the messages.

Appendix 22 – BAN logic Verification WiMAX PKMv2 Key Exchange

SS - Subscriber Station

BS - Base Station

K - shared key

A - Authentication

Initial Assumptions:

$SS \mid\equiv \overset{K_A}{\longmapsto} A$

$BS \mid\equiv \overset{K_A}{\longmapsto} A$

$SS \mid\equiv \overset{K_{SS}}{\longmapsto} SS$

$BS \mid\equiv \overset{K_{SS}}{\longmapsto} SS$

$SS \mid\equiv A \Rightarrow \overset{K}{\longmapsto} BS$

$BS \mid\equiv A \Rightarrow \overset{K}{\longmapsto} SS$

$SS \mid\equiv BS \Rightarrow (SS \overset{K}{\leftrightarrow} BS)$

$BS \mid\equiv SS \overset{K}{\leftrightarrow} BS$

$SS \mid\equiv \#(T_{ABS})$

$SS \mid\equiv \#(N_{SS})$

$SS \mid\equiv \#(T_{BS})$

$BS \mid\equiv \#(T_{A_{SS}})$

$BS \mid\equiv \#(N_{BS})$

$BS \mid\equiv \#(T_{SS})$

$BS \mid\equiv \overset{K_{SS}}{\longmapsto} SS$

$SS \mid\equiv \overset{K_{BS}}{\longmapsto} BS$

Idealized Protocol:

Message 1: $BS \triangleleft SS.REQ, \{T_{A_{SS}}, \overset{K_{SS}}{\longmapsto} SS\}_{K_{A^{-1}}}$

Message 2: $SS \triangleleft \{N_{SS}, N_{BS}, \{SS \overset{AK}{\longleftrightarrow} BS\}_{K_{SS}}\}_{K_{BS^{-1}}}, \{T_{ABS}, \overset{K_{BS}}{\longmapsto} BS\}_{K_{A^{-1}}}$

Message 3: $BS \triangleleft \{N_{BS}, SS.RPL\}_{AK}$

Protocol Goal:

$SS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

$BS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

$BS \mid\equiv SS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

$SS \mid\equiv BS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

Verification:

$SS \triangleleft BS \to SS : \{N_{SS}, N_{BS}, \{SS \overset{AK}{\longleftrightarrow} BS\}_{K_{SS}}\}_{K_{BS^{-1}}}, \{T_{ABS}, \overset{K_{BS}}{\longmapsto} BS\}_{K_{A^{-1}}}$ and with initial assumption of $SS \mid\equiv \overset{K_{BS}}{\longmapsto} BS$ it can be deduced by applying message-meaning rule that:

$SS \mid\equiv BS \mid\sim (N_{SS}, N_{BS}, \{SS \overset{AK}{\longleftrightarrow} BS\}_{K_{SS}})$

With last message and initial assumption of $SS \mid\equiv \#(N_{SS})$ it can be deduced with nonce verification that:

$SS \mid\equiv BS \mid\equiv \{SS \overset{AK}{\longleftrightarrow} BS\}_{K_{SS}}$

Because the signing of the key, it can be assumed that:

$SS \mid\equiv BS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

Next, by applying jurisdiction rule to last message with initial assumption of $SS \mid\equiv BS \Rightarrow (SS \overset{K}{\leftrightarrow} BS)$:

$SS \mid\equiv SS \overset{AK}{\longleftrightarrow} BS$

Next, $BS \triangleleft SS \to BS : \{N_{BS}, SS.RPL\}_{AK}$ and with the initial assumptions of $BS \mid\equiv SS \overset{K}{\leftrightarrow} BS$ and $BS \mid\equiv \#(N_{BS})$ it can be deduced that

$BS \mid\equiv SS.RPL$

The authentication comes through, but BS cannot verify if the authentication process was started by SS, because the identity was not included in the requesting messages.

Appendix 23 – BAN logic Verification for GSM Key Exchange

Initial Assumptions:

$MSC \mid\equiv MS \mid\sim (ServiceRequest)$

$VLR \mid\equiv MSC \mid\sim (IMSI)$

$HLR \mid\equiv VLR \mid\sim (IMSI)$

$VLR \mid\equiv MSC \mid\sim (IMSI)$

$VLR \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$

$MS \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$

Idealized Protocol:

$VLR \triangleleft \{Response2\}_{EncryptionKey}$

Protocol Goal:

$MS \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$

$VLR \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$

$VLR \mid\equiv MS \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$

$MS \mid\equiv VLR \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$

Verification:

With the communication being initiated by MS with sending ServiceRequest to MSC, which includes CKSN, IMSI, TMSI and is forwarded to VLR, who sends Authentication-Request(IMSI) to HLR. With the initial

belief of that $HLR \mid\equiv MS \mid\sim IMSI$, it sends out IMSI, generates RAND, with that K_C, which is used to generate SRES to VLR. VLR starts the authentication process, with sending out

CKSN and RAND which is forwarded to MS. MS then uses its private key to calculate SRES and sends a response containing SRES to MSC, who sends it to VLR, who checks if SRES matches with

SRES received from HLR. If it matches, MSC receives the response with success in it, which is forwarded to MS. Then MS is authenticated and can start communication with EncryptionKey, which

is generated with MS private key and A5 algorithm.

From fifth message, it can be deduced that :

After recieving Response from MSC, MS generates EncryptionKey and $MS \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$, because of the symmetry of the key, $MS \mid\equiv VLR \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$.

While recieving the message $\{Response2\}_{EncryptionKey}$ and with initial belief that $VLR \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$, it can be deduced, that

$VLR \triangleleft Response2$ and because of that $VLR \mid\equiv MS \mid\equiv MS \xleftrightarrow{EncryptionKey} VLR$.

Appendix 24 – BAN logic Verification for EAP-AKA Key Exchange [1]

U - User HE - Home Environment SN - Serving Network r - random number K - shared key SEQ - sequence number for synchronization between HE and U RES = f2(K, r) = expected response CK = f3(K, r) = cipher key IK = f4(K, r) = integrity key K_a= f5(K, r) = anonymity key AUTN = authentication token

Idealized Protocol:

$SN \triangleleft r, f2(K,r), f3(K,r), f4(K,r), \{SEQ\}_{f5(K,r)}, f1(K,SEQ,r)$

$U \triangleleft r, \{SEQ\}_{f5(K,r)}, f1(K,SEQ,r)$

$SN \triangleleft f2(K,r)$

Initial Assumptions:

$SN \mid\equiv HE \Rightarrow (SN \overset{K'}{\longleftrightarrow} U, \#(K'))$

$SN \mid\equiv (HE \mid\sim RES \rightarrow HE \mid\equiv (SN \overset{RES}{\longleftrightarrow} U, \#(RES)))$

$SN \mid\equiv (HE \mid\sim CK \rightarrow HE \mid\equiv (SN \overset{CK}{\longleftrightarrow} U, \#(CK)))$

$SN \mid\equiv (HE \mid\sim IK \rightarrow HE \mid\equiv (SN \overset{IK}{\longleftrightarrow} U, \#(IK)))$

$SN \mid\equiv HE \mid\sim (r, RES, CK, IK, AUTN)$

$SN \mid\equiv (U \mid\sim RES \rightarrow U \triangleleft (CK, IK))$

$SN \mid\equiv !(SN \mid\sim RES)$

$U \triangleleft K$

$U \mid\equiv HE \overset{K}{\leftrightarrow} U$

$U \mid\equiv HE \triangleleft K$

$U \mid\equiv \#(SEQ)$

$U \mid\equiv !(U \mid\sim f1(K,r))$

$U \mid\equiv HE \Rightarrow SN \overset{K'}{\longleftrightarrow} U$

$U \mid\equiv (HE \mid\sim r \rightarrow HE \mid\equiv SN \overset{fi(K,r)}{\longleftrightarrow} U)$ for i=3,4

$U \mid\equiv HE \Rightarrow \#(r)$

$U \mid\equiv (HE \mid\sim r \rightarrow HE \mid\equiv \#(r))$

$HE \triangleleft K$

$HE \mid\equiv HE \overset{K}{\leftrightarrow} U$

Protocol Goal:

$SN \mid\equiv U \mid\sim RES$

$U \mid\equiv HE \mid\sim (SEQ, r)$

$SN \triangleleft CK$

$SN \triangleleft IK$

$U \triangleleft f3(K, r)$

$U \triangleleft f4(K, r)$

$SN \mid\equiv SN \xleftrightarrow{CK} U$

$SN \mid\equiv SN \xleftrightarrow{IK} U$

$U \mid\equiv SN \xleftrightarrow{f3(K,r)} U$

$U \mid\equiv SN \xleftrightarrow{f4(K,r)} U$

$SN \mid\equiv \#(CK)$

$SN \mid\equiv \#(IK)$

$U \mid\equiv \#(f3(K, r))$

$U \mid\equiv \#(f4(K, r))$

$SN \mid\equiv U \triangleleft (CK, IK)$

$U \mid\equiv HE \triangleleft (f3(K, r), f4(K, r))$

$HE \mid\equiv HE \xleftrightarrow{f5(K,r)} U$

Verification:

Receiving first message, $SN \triangleleft (CK, IK)$, goal

From initial assumptions of $SN \mid\equiv HE \mid\sim (r, RES, CK, IK, AUTN)$ and $SN \mid\equiv (HE \mid\sim RES \rightarrow HE \mid\equiv (SN \xleftrightarrow{RES} U, \#(RES)))$, because of the interference and rationality rule, it can be deduced that

$SN \mid\equiv HE \mid\equiv (SN \xleftrightarrow{RES} U, \#(RES))$

From last deduction and initial assumption of $SN \mid\equiv HE \Rightarrow (SN \xleftrightarrow{K'} U, \#(K'))$:

$SN \mid\equiv (SN \xleftrightarrow{RES} U, \#(RES))$

Now it can be proved that :

$SN \mid\equiv SN \xleftrightarrow{CK} U$

$SN \mid\equiv SN \xleftrightarrow{IK} U$

$SN \mid\equiv \#(CK)$

$SN \mid\equiv \#(IK)$

From second message, $U \triangleleft r$ and because of that and initial assumption of $U \triangleleft K$, it can be proved that $U \triangleleft K, r$, therefor $U \triangleleft fi(K, r)$ for i=2,..,5 ; with this deduction $U \triangleleft SEQ$.

Because of these deductions and second message, it can be shown that $U \mid\equiv U \triangleleft f1(K, SEQ, r)$, because U is able to calculate it.

With initial assumptions of $U \mid\equiv HE \xleftrightarrow{K} U$ and $U \mid\equiv !(U \mid\sim f1(K, r))$, and the last calculation and the initial assumption of $U \mid\equiv \#(SEQ)$, it can be shown that $U \mid\equiv HE \mid\sim (SEQ, r)$.

In result of last deduction $U \mid\equiv HE \triangleleft fi(K, r)$, for i = 3,4.

Using initial assumption of $U \mid\equiv (HE \mid\sim r \rightarrow HE \mid\equiv SN \xleftrightarrow{fi(K,r)} U)$ for i=3,4 and $U \mid\equiv HE \mid\sim (SEQ, r)$ it can be deduced that $U \mid\equiv (HE \mid\equiv SN \xleftrightarrow{fi(K,r)} U)$ for i= 3,4.

With last message and initial assumption of $U \mid\equiv HE \Rightarrow SN \overset{K'}{\leftrightarrow} U$ that $U \mid\equiv SN \xleftrightarrow{fi(K,r)U}$ for i = 3,4.

Using initial assumption of $U \mid\equiv (HE \mid\sim r \rightarrow HE \mid\equiv \#(r))$ it can be said, that $U \mid\equiv HE \mid\equiv \#(r)$ and with this $U \mid\equiv \#(r)$ therefore $U \mid\equiv \#(fi(K,r))$ for i=3,4.

From third message and initial assumption of received expected response from $SN \mid\equiv HE \mid\sim (r, RES, CK, IK, AUTN)$

$SN \mid\equiv SN \triangleleft RES$

$SN \mid\equiv U \mid\sim RES$

$SN \mid\equiv U \triangleleft (CK, IK)$

With initial assumption of $HE \mid\equiv HE \overset{K}{\leftrightarrow} U$ and previous beliefs of fi(K,r) for i= 2,..,5 that

$HE \mid\equiv HE \xleftrightarrow{f5(K,r)U}$

Appendix 25 – BAN logic Verification for ZigBee Key Exchange

Initial Assumptions:

$TrustCenter \mid\equiv TC$

$TrustCenter \mid\equiv Joiner \mid\equiv J$

$TrustCenter \mid\equiv Joiner \mid\equiv aQ$

$TrustCenter \mid\equiv sub-MAC$

$Joiner \mid\equiv J$

$Joiner \mid\equiv TrustCenter \mid\equiv TC$

$Joiner \mid\equiv TrustCenter \mid\equiv bQ$

$Joiner \mid\equiv sub-MAC$

Idealized Protocol:

$Joiner \triangleleft J, aQ, N_S, sub-MAC(aQ, N_S, J)$

$TrustCenter \triangleleft TC, bQ, N_{S+1}, sub-MAC(K)$

$Joiner \triangleleft E_K, (N_{S+1})$

Protocol Goal:

$TrustCenter \mid\equiv TrustCenter \overset{K}{\leftrightarrow} Joiner$

$Joiner \mid\equiv TrustCenter \overset{K}{\leftrightarrow} Joiner$

$Joiner \mid\equiv TrustCenter \mid\equiv TrustCenter \overset{K}{\leftrightarrow} Joiner$

$TrustCenter \mid\equiv Joiner \mid\equiv TrustCenter \overset{K}{\leftrightarrow} Joiner$

Verification:

From first idealized message, it can be deduced that $\frac{Joiner \triangleleft J, aQ, N_S, sub-MAC(aQ, N_S, J}{Joiner \mid\equiv J, aQ, N_S}$
and

therefor because $\frac{Joiner \mid\equiv aQ}{Joiner \mid\equiv K}$

From second idealized message, $TrustCenter \triangleleft (TC, bQ, N_{S+1})$, TrustCenter will compute K and then sub-MAC(K).

Because TrustCenter calculates K and sub-MAC(K), it can be deduced that

$\frac{TrustCenter \triangleleft sub-MAC(K), TrustCenter \mid\equiv sub-MAC}{TrustCenter \mid\equiv K}$,

next $\frac{TrustCenter \mid\equiv K}{TrustCenter \mid\equiv bQ, TrustCenter \mid\equiv \#(N_{S+1})}$ and finally

$\frac{TrustCenter \mid\equiv K}{TrustCenter \mid\equiv TrustCenter \overset{K}{\leftrightarrow} Joiner}$.

Next, $TrustCenter \mid\equiv Joiner \mid\equiv TrustCenter \overset{k}{\leftrightarrow} Joiner$, because of the Symmetric-Key Key Establishment (SKKE) protocol.

From last idealized message, $\dfrac{Joiner \triangleleft \{N_{S+1}\}_K}{Joiner \mid\equiv TrustCenter \overset{K}{\leftrightarrow} Joiner}$.

Because of the last message, it can be said that $Joiner \mid\equiv TrustCenter \mid\equiv TrustCenter \overset{K}{\leftrightarrow} Joiner$, because of the SKKE protocol.

Appendix 26 – BAN logic Verification for PSK Key Exchange

Initial Assumptions:

$Device fresh AT_RAND_{AP}$

$AP\#(AT_RAND_D)$

Idealized Protocol:

$Device \triangleleft AT_RAND_{AP}$

$AP \triangleleft AT_RAND_D, AT_MAC_D$

$Device \triangleleft \{AT_MAC_{AP}\}_{TEK}$

$AP \triangleleft \{AT_MAC_D\}_{TEK}$

$Device \triangleleft \{EAPSuccess\}_{TEK}$

Protocol Goal:

$Device \mid\equiv Device \overset{TEK}{\longleftrightarrow} AP$

$Device \mid\equiv AP \mid\equiv Device \overset{TEK}{\longleftrightarrow} AP$

$AP \mid\equiv Device \overset{TEK}{\longleftrightarrow} AP$

$AP \mid\equiv Device \mid\equiv Device \overset{TEK}{\longleftrightarrow} AP$

Verification:

While AP sends out AT_RAND_{AP}, which was generated by itself, therefore $AP \mid\equiv \#(AT_RAND_{AP})$.

From first message $Device \triangleleft AT_RAND_{AP}$, next device generated its own random and sends it to AP while adding message authentication code in the second message$AP \triangleleft AT_RAND_D, AT_MAC_D$.

Because the device generated AT_RAND_D, therefore $Device \mid\equiv \#(AT_RAND_D)$.

From second message, AP check validity of the AT_MAC_D. If it is valid, then $AP \mid\equiv AT_MAC_D$, it can be deduced that $AP \mid\equiv Device \mid\sim AT_RAND_D$ as well.

AP sends out message $\{AT_MAC_{AP}\}_{TEK}$, which Device should be able to derive.

With third message, because of Device is able to resolve the challenge TEK with its pre-shared key $Device \triangleleft AT_MAC_{AP}$. With the validity of AT_MAC_{AP} then $Device \mid\equiv AT_MAC_{AP}$, $Device \mid\equiv AP \mid\sim \{AT_MAC_{AP}\}_{TEK}$.

Also, then $Device \mid\equiv Device \overset{TEK}{\longleftrightarrow} AP$.

With fourth message, $AP \triangleleft \{AT_MAC_D\}_{TEK}$, while AP is able to resolve the message using its key, $AP \mid\equiv Device \overset{TEK}{\longleftrightarrow} AP$ for communication and $AP \mid\equiv Device \mid\equiv Device \overset{TEK}{\longleftrightarrow} AP$, because of the response.

With fifth message $Device \triangleleft \{EAPSuccess\}_{TEK}$, Device is able to deduct that $Device \mid\equiv AP \mid\equiv Device \overset{TEK}{\longleftrightarrow} AP$.

Appendix 27 – BAN logic Verification for CCM and CMAC Key Exchange

Since CMAC and CCM BAN logic results and assumptions were the same, they have been combined into one appendix. Difference is that CMAC encryption is applied to whole message and CCM or CBC-MAC last block of the message.

Assumptions:

$$A \mid\equiv \#(T_1)$$

$$B \mid\equiv \#(T_2)$$

$$A \mid\equiv A \overset{K}{\leftrightarrow} B$$

$$B \mid\equiv A \overset{K}{\leftrightarrow} B$$

Idealizad Protocol:

$$A \triangleleft \{I_1, T_1\}_K$$

$$B \triangleleft \{I_2, T_2\}_K$$

Goal:

$$A \mid\equiv B \mid\sim I_1$$

$$B \mid\equiv A \mid\sim I_2$$

Verification:

With initial assumption of that

$$A \mid\equiv A \overset{K}{\leftrightarrow} B$$

, then A receives first message, this is deduced with message-meaning rule:

$$\frac{A \mid\equiv A \overset{K}{\leftrightarrow} B, A\triangleleft}{\{I_1, T_1\}_K}$$

.

Applying shared secret rule results with the initial assumption of

$$A \mid\equiv A \overset{K}{\leftrightarrow} B$$

results :

$$\frac{A \triangleleft \{I_1, T_1\}_K, A \mid\equiv A \overset{K}{\leftrightarrow} B}{A \triangleleft I_1, T_1}$$

.

Applying message-meaning rule,

$$\frac{A \mid\equiv A \overset{K}{\leftrightarrow} B, A \triangleleft I_1, T_1}{A \mid\equiv B \mid\sim I_1}$$

.

After A was able to read the message,

A will respond with its own identity and same thing applies to

$$B \mid\equiv A \overset{K}{\leftrightarrow} B$$

, this is also deduced with message-meaning rule:

$$\frac{B \mid\equiv A \overset{K}{\leftrightarrow} B, B\triangleleft}{\{I_2, T_2\}_K}$$

.

Applying shared secret rule results with the initial assumption of

$$B \mid\equiv A \overset{K}{\leftrightarrow} B$$

results :

$$\frac{B \triangleleft \{I_2, T_2\}_K, B \mid\equiv A \overset{K}{\leftrightarrow} B}{B \triangleleft I_2, T_2}$$

.

Applying message-meaning rule,

$$\frac{B \mid\equiv A \overset{K}{\leftrightarrow} B, B \triangleleft I_2, T_2}{B \mid\equiv A \mid\sim I_2}$$

.

Appendix 28 – Used Terms for Literature Review

Term searched	Total results	Used works	Date	Source
Smart Home protocols	534,000 results	5	11/09/2019	Google Scholar
Wireless protocols	947,000 restults	4	11/09/2019	Google Scholar
infrared communication protocol	77 results	2	11/09/2019	Google Scholar
infrared protocol for smart home	50,000 results	2	11/09/2019	Google Scholar
infrared protocols for authentication	22,500 results	0	11/09/2019	Google Scholar
infrared protocol authentication	26,200 results	0	11/09/2019	Google Scholar
infrared protocol authentication in smart home networks	15,100 results	0	11/09/2019	Google Scholar
infrared protocol authentication with IOT devices	21,000 results	0	11/09/2019	Google Scholar
infrared protocol design	527,000 results	0	11/09/2019	Google Scholar
infrared protocol	1,370,000 results	1	11/09/2019	Google Scholar
infrared protocol comparison	875,000 results	1	11/09/2019	Google Scholar
infrared protocol comparison for authentication	26,000 results	0	11/09/2019	Google Scholar
infrared protocol standard	814,000 results	0	11/09/2019	Google Scholar
wifi communication protocol	73,100 results	7	11/09/2019	Google Scholar
wifi protocol for smart home	32,600 results	4	11/09/2019	Google Scholar
wifi protocols for authentication	27400 results	3	11/09/2019	Google Scholar
wifi protocol authentication	27100 results	1	11/09/2019	Google Scholar
wifi protocol authentication in smart home networks	15,700 results	5	11/09/2019	Google Scholar
wifi protocol authentication with IOT devices	23,800 results	6	11/09/2019	Google Scholar
wifi protocol design	56100 results	0	11/09/2019	Google Scholar
wifi protocol	69,800 results	1	11/09/2019	Google Scholar
wifi protocol comparison	45,500 results	3	11/09/2019	Google Scholar
wifi protocol comparison for authentication	19200 results	1	11/09/2019	Google Scholar
wifi protocol standard	54600 results	0	11/09/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
bluetooth communication protocol	102,000 results	6	11/09/2019	Google Scholar
bluetooth protocol for smart home	39,100 results	4	11/09/2019	Google Scholar
bluetooth protocols for authentication	35,000 results	0	11/09/2019	Google Scholar
bluetooth protocol authentication	34,700 results	1	11/09/2019	Google Scholar
bluetooth protocol authentication in smart home networks	19,700 results	1	11/09/2019	Google Scholar
bluetooth protocol authentication with IOT devices	23,500 results	0	11/09/2019	Google Scholar
bluetooth protocol design	78,000 results	0	11/09/2019	Google Scholar
bluetooth protocol	92,000 results	0	11/09/2019	Google Scholar
bluetooth protocol comparison	53,300 results	2	11/09/2019	Google Scholar
bluetooth protocol comparison for authentication	23,300 results	0	11/09/2019	Google Scholar
bluetooth protocol standard	70,700 results	1	09/12/2019	Google Scholar
thread protocol for iot	11,800 results	1	09/12/2019	Google Scholar
thread protocol for smart home	47,900 results	1	09/12/2019	Google Scholar
thread protocol for authentication	70,300 results	0	09/12/2019	Google Scholar
thread protocol authentication	65,900 results	0	09/12/2019	Google Scholar
thread protocol authentication in smart home networks	23,500 results	0	09/12/2019	Google Scholar
thread protocol authentication with IOT devices	6,260 results	1	09/12/2019	Google Scholar
thread protocol design	292,000 results	0	09/12/2019	Google Scholar
thread protocol	392,000 results	1	09/12/2019	Google Scholar
thread protocol comparison	214,000 results	0	09/12/2019	Google Scholar
thread protocol comparison for authentication	40,000 results	0	09/12/2019	Google Scholar
thread protocol standard	149,000 results	1	09/12/2019	Google Scholar
Zigbee communication protocol	64,200 results	0	09/12/2019	Google Scholar
Zigbee protocol for smart home	31,900 results	3	09/12/2019	Google Scholar
Zigbee protocols for authentication	18,500 results	0	09/12/2019	Google Scholar
Zigbee protocol authentication	18,700 results	1	09/12/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
Zigbee protocol authentication in smart home networks	14,200 results	1	09/12/2019	Google Scholar
Zigbee protocol authentication with IOT devices	17,800 results	0	09/12/2019	Google Scholar
Zigbee protocol design	56,800 results	0	09/12/2019	Google Scholar
Zigbee protocol	62,200 results	6	09/12/2019	Google Scholar
Zigbee protocol comparison	38,100 results	1	09/12/2019	Google Scholar
Zigbee protocol comparison for authentication	11,500 results	0	09/12/2019	Google Scholar
Zigbee protocol standard	51,900 results	0	09/12/2019	Google Scholar
Z-Wave communication protocol	5,130 results	0	09/12/2019	Google Scholar
Z-Wave protocol for smart home	4,390 results	2	09/12/2019	Google Scholar
Z-Wave protocols for authentication	2,000 results	0	09/12/2019	Google Scholar
Z-Wave protocol authentication	2,010 results	1	09/12/2019	Google Scholar
Z-Wave protocol authentication in smart home networks	1,860 results	1	09/12/2019	Google Scholar
Z-Wave protocol authentication with IOT devices	1,960 results	1	09/12/2019	Google Scholar
Z-Wave protocol design	4,830 results	1	09/12/2019	Google Scholar
Z-Wave protocol	5,450 results	3	09/12/2019	Google Scholar
Z-Wave protocol comparison	2,950 results	1	09/12/2019	Google Scholar
Z-Wave protocol comparison for authentication	1,570 results	0	09/12/2019	Google Scholar
Z-Wave protocol standard	4,700 results	0	09/12/2019	Google Scholar
WeMo communication protocol	653 results	0	09/18/2019	Google Scholar
WeMo protocol for smart home	564 results	0	09/18/2019	Google Scholar
WeMo protocols for authentication	297 results	0	09/18/2019	Google Scholar
WeMo protocol authentication	297 results	0	09/18/2019	Google Scholar
WeMo protocol authentication in smart home networks	283 results	0	09/18/2019	Google Scholar
WeMo protocol authentication with IOT devices	295 results	0	09/18/2019	Google Scholar
WeMo protocol design	580 results	0	09/18/2019	Google Scholar
WeMo protocol	763 results	0	09/18/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
WeMo protocol comparison	468 results	0	09/18/2019	Google Scholar
WeMo protocol comparison for authentication	190 results	0	09/18/2019	Google Scholar
WeMo protocol standard	614 results	0	09/18/2019	Google Scholar
Wired Protocols for smart homes	29,300 results	4	09/18/2019	Google Scholar
Wireless protocols for Smart Home	157,000 results	8	09/18/2019	Google Scholar
Hybrid protocols for smart home	90,800 results	5	09/18/2019	Google Scholar
PLC BUS communication protocol	78,700 results	1	09/18/2019	Google Scholar
PLC BUS protocol for smart home	22,600 results	1	09/18/2019	Google Scholar
PLC BUS protocols for authentication	19,200 results	0	09/18/2019	Google Scholar
PLC BUS protocol authentication	21,400 results	0	09/18/2019	Google Scholar
PLC BUS protocol authentication in smart home networks	15,600 results	0	09/18/2019	Google Scholar
PLC BUS protocol authentication with IOT devices	15,600 results	0	09/18/2019	Google Scholar
PLC BUS protocol design	59,800 results	0	09/18/2019	Google Scholar
PLC BUS protocol	74,800 results	3	09/18/2019	Google Scholar
PLC BUS protocol comparison	33,200 results	1	09/18/2019	Google Scholar
PLC BUS protocol comparison for authentication	15,100 results	0	09/18/2019	Google Scholar
PLC BUS protocol standard	62,100 results	1	09/18/2019	Google Scholar
UPB communication protocol	4,380 results	0	09/18/2019	Google Scholar
UPB protocol for smart home	578 results	2	09/18/2019	Google Scholar
UPB protocols for authentication	661 results	0	09/18/2019	Google Scholar
UPB protocol authentication	661 results	0	09/18/2019	Google Scholar
UPB protocol authentication in smart home networks	338 results	0	09/18/2019	Google Scholar
UPB protocol authentication with IOT devices	576 results	0	09/18/2019	Google Scholar
UPB protocol design	4,670 results	0	09/18/2019	Google Scholar
UPB protocol	6,420 results	4	09/18/2019	Google Scholar
UPB protocol comparison	3,670 results	1	09/18/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
UPB protocol comparison for authentication	353 resultsx	0	09/18/2019	Google Scholar
UPB protocol standard	4,510 results	2	09/18/2019	Google Scholar
KNX communication protocol	3,630 results	4	09/18/2019	Google Scholar
KNX protocol for smart home	2,600 results	4	09/18/2019	Google Scholar
KNX protocols for authentication	921 results	3	09/18/2019	Google Scholar
KNX protocol authentication	921 results	2	09/18/2019	Google Scholar
KNX protocol authentication in smart home networks	789 results	0	09/18/2019	Google Scholar
KNX protocol authentication with IOT devices	865 results	0	09/18/2019	Google Scholar
KNX protocol design	3,500 results	0	09/18/2019	Google Scholar
KNX protocol	3,870 results	1	09/18/2019	Google Scholar
KNX protocol comparison	1,900 results	0	09/18/2019	Google Scholar
KNX protocol comparison for authentication	521 results	1	09/18/2019	Google Scholar
KNX protocol standard	3,350 results	2	09/18/2019	Google Scholar
INSTEON communication protocol	1,060 results	0	09/18/2019	Google Scholar
INSTEON protocol for smart home	956 results	3	09/18/2019	Google Scholar
INSTEON protocols for authentication	368 results	0	09/18/2019	Google Scholar
INSTEON protocol authentication	368 results	0	09/18/2019	Google Scholar
INSTEON protocol authentication in smart home networks	368 results	0	09/18/2019	Google Scholar
INSTEON protocol authentication with IOT devices	378 results	0	09/18/2019	Google Scholar
INSTEON protocol design	1,020 results	2	09/18/2019	Google Scholar
INSTEON protocol	1,030 results	2	09/18/2019	Google Scholar
INSTEON protocol comparison	641 results	1	09/18/2019	Google Scholar
INSTEON protocol comparison for authentication	264 results	0	09/18/2019	Google Scholar
INSTEON protocol standard	992 results	0	09/18/2019	Google Scholar
6LOWPAN communication protocol	18,900 results	2	09/22/2019	Google Scholar
6LOWPAN protocol for smart home	11,000 results	3	09/22/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
6LOWPAN protocols for authentication	8,320 results	0	09/22/2019	Google Scholar
6LOWPAN protocol authentication	8,360 results	0	09/22/2019	Google Scholar
6LOWPAN protocol authentication in smart home networks	6,290 results	2	09/22/2019	Google Scholar
6LOWPAN protocol authentication with IOT devices	8,160 results	1	09/22/2019	Google Scholar
6LOWPAN protocol design	16,500 results	0	09/22/2019	Google Scholar
6LOWPAN protocol	19,800 results	1	09/22/2019	Google Scholar
6LOWPAN protocol comparison	10,700 results	0	09/22/2019	Google Scholar
6LOWPAN protocol comparison for authentication	4,990 results	0	09/22/2019	Google Scholar
6LOWPAN protocol standard	16,400 results	0	09/22/2019	Google Scholar
IrDA communication protocol	36,300 results	2	09/22/2019	Google Scholar
IrDA protocol for smart home	17,100 results	1	09/22/2019	Google Scholar
IrDA protocols for authentication	17,600 results	0	09/22/2019	Google Scholar
IrDA protocol authentication	18,100 results	0	09/22/2019	Google Scholar
IrDA protocol authentication in smart home networks	13,400 results	0	09/22/2019	Google Scholar
IrDA protocol authentication with IOT devices	13,000 results	0	09/22/2019	Google Scholar
IrDA protocol design	25,200 results	1	09/22/2019	Google Scholar
IrDA protocol	33,900 results	3	09/22/2019	Google Scholar
IrDA protocol comparison	18,900 results	2	09/22/2019	Google Scholar
IrDA protocol comparison for authentication	9,970 results	0	09/22/2019	Google Scholar
IrDA protocol standard	29,900 results	1	09/22/2019	Google Scholar
Cellular communication protocol	2,960,000 results	0	09/22/2019	Google Scholar
Cellular protocol for smart home	272,000 results	5	09/22/2019	Google Scholar
Cellular protocols for authentication	303,000 results	0	09/22/2019	Google Scholar
Cellular protocol authentication	319,000 results	1	09/22/2019	Google Scholar
Cellular protocol authentication in smart home networks	97,700 results	0	09/22/2019	Google Scholar
Cellular protocol authentication with IOT devices	40,000 results	0	09/22/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
Cellular protocol design	2,640,000 results	2	09/22/2019	Google Scholar
Cellular protocol	3,250,000 results	0	09/22/2019	Google Scholar
Cellular protocol comparison	3,300,000 results	2	09/22/2019	Google Scholar
Cellular protocol comparison for authentication	113,000 results	2	09/22/2019	Google Scholar
Cellular protocol standard	2,690,000 results	0	09/22/2019	Google Scholar
3G communication protocol	212,000 results	1	09/22/2019	Google Scholar
3G protocol for smart home	66,000 results	3	09/22/2019	Google Scholar
3G protocols for authentication	88,400 results	1	09/22/2019	Google Scholar
3G protocol authentication	90,900 results	1	09/22/2019	Google Scholar
3G protocol authentication in smart home networks	39,500 results	0	09/22/2019	Google Scholar
3G protocol authentication with IOT devices	27,500 results	0	09/22/2019	Google Scholar
3G protocol design	205,000 results	0	09/22/2019	Google Scholar
3G protocol	318,000 results	1	09/22/2019	Google Scholar
3G protocol comparison	193,000 results	1	09/22/2019	Google Scholar
3G protocol comparison for authentication	41,600 results	0	09/22/2019	Google Scholar
3G protocol standard	273,000 results	0	09/22/2019	Google Scholar
4G communication protocol	152,000 results	0	09/22/2019	Google Scholar
4G protocol for smart home	48,100 results	2	09/22/2019	Google Scholar
4G protocols for authentication	53,400 results	0	09/22/2019	Google Scholar
4G protocol authentication	54,600 results	1	09/22/2019	Google Scholar
4G protocol authentication in smart home networks	29,800 results	0	09/22/2019	Google Scholar
4G protocol authentication with IOT devices	27,400 results	2	09/22/2019	Google Scholar
4G protocol design	134,000 results	0	09/22/2019	Google Scholar
4G protocol	236,000 results	1	09/22/2019	Google Scholar
4G protocol comparison	142,000 results	0	09/22/2019	Google Scholar
4G protocol comparison for authentication	29,900 results	0	09/22/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
4G protocol standard	164,000 results	0	09/22/2019	Google Scholar
2G communication protocol	113,000 results	2	09/22/2019	Google Scholar
2G protocol for smart home	27,300 results	1	09/22/2019	Google Scholar
2G protocols for authentication	31,600 results	1	09/22/2019	Google Scholar
2G protocol authentication	34,000 results	1	09/22/2019	Google Scholar
2G protocol authentication in smart home networks	22,200 results	0	09/22/2019	Google Scholar
2G protocol authentication with IOT devices	19,300 results	0	09/22/2019	Google Scholar
2G protocol design	125,000 results	0	09/22/2019	Google Scholar
2G protocol	214,000 results	1	09/22/2019	Google Scholar
2G protocol comparison	157,000 results	0	09/22/2019	Google Scholar
2G protocol comparison for authentication	19,200 results	0	09/22/2019	Google Scholar
2G protocol standard	167,000 results	0	09/22/2019	Google Scholar
Radio Frequency communication protocol	2,180,000 results	5	09/22/2019	Google Scholar
Radio Frequency protocol for smart home	371,000 results	4	09/22/2019	Google Scholar
Radio Frequency protocols for authentication	401,000 results	2	09/22/2019	Google Scholar
Radio Frequency protocol authentication	440,000 results	3	09/22/2019	Google Scholar
Radio Frequency protocol authentication in smart home networks	186,000 results	2	09/22/2019	Google Scholar
Radio Frequency protocol authentication with IOT devices	42,300 results	2	09/22/2019	Google Scholar
Radio Frequency protocol design	1,940,000 results	1	09/22/2019	Google Scholar
Radio Frequency protocol	2,750,000 results	0	09/22/2019	Google Scholar
Radio Frequency protocol comparison	1,620,000 results	0	09/22/2019	Google Scholar
Radio Frequency protocol comparison for authentication	189,000 results	0	09/22/2019	Google Scholar
Radio Frequency protocol standard	1,850,000 results	0	09/22/2019	Google Scholar
Ethernet communication protocol	652,000 results	4	09/22/2019	Google Scholar
Ethernet protocol for smart home	111,000 results	3	09/22/2019	Google Scholar
Ethernet protocols for authentication	198,000 results	5	09/22/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
Ethernet protocol authentication	206,000 results	4	09/22/2019	Google Scholar
Ethernet protocol authentication in smart home networks	82,300 results	1	09/22/2019	Google Scholar
Ethernet protocol authentication with IOT devices	30,400 results	0	09/22/2019	Google Scholar
Ethernet protocol design	419,000 results	0	09/22/2019	Google Scholar
Ethernet protocol	705,000 results	2	09/22/2019	Google Scholar
Ethernet protocol comparison	238,000 results	6	09/22/2019	Google Scholar
Ethernet protocol comparison for authentication	81,600 results	2	09/22/2019	Google Scholar
Ethernet protocol standard	503,000 results	1	09/22/2019	Google Scholar
X10 communication protocol	30,600 results	5	09/22/2019	Google Scholar
X10 protocol for smart home	7,270 results	4	09/22/2019	Google Scholar
X10 protocols for authentication	6,010 results	1	09/22/2019	Google Scholar
X10 protocol authentication	6,000 results	1	09/22/2019	Google Scholar
X10 protocol authentication in smart home networks	6,680 results	1	09/22/2019	Google Scholar
X10 protocol authentication with IOT devices	11,100 results	0	09/22/2019	Google Scholar
X10 protocol design	42,700 results	0	09/22/2019	Google Scholar
X10 protocol	58,500 results	3	09/22/2019	Google Scholar
X10 protocol comparison	50,100 results	0	09/22/2019	Google Scholar
X10 protocol comparison for authentication	4,040 results	0	09/22/2019	Google Scholar
X10 protocol standard	49,300 results	0	09/22/2019	Google Scholar
Ultra Wide Band communication protocol	160,000 results	0	09/22/2019	Google Scholar
Ultra Wide Band protocol for smart home	39,000 results	1	09/22/2019	Google Scholar
Ultra Wide Band protocols for authentication	31,800 results	0	09/22/2019	Google Scholar
Ultra Wide Band protocol authentication	34,400 results	0	09/22/2019	Google Scholar
Ultra Wide Band protocol authentication in smart home networks	28,200 results	0	09/22/2019	Google Scholar
Ultra Wide Band protocol authentication with IOT devices	22,000 results	0	09/22/2019	Google Scholar
Ultra Wide Band protocol design	180,000 results	0	09/22/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
Ultra Wide Band protocol	240,000 results	2	09/22/2019	Google Scholar
Ultra Wide Band protocol comparison	172,000 results	1	09/22/2019	Google Scholar
Ultra Wide Band protocol comparison for authentication	24,200 results	0	09/22/2019	Google Scholar
Ultra Wide Band protocol standard	202,000 results	0	09/22/2019	Google Scholar
HART communication protocol	179,000 results	3	09/22/2019	Google Scholar
HART protocol for smart home	29,700 results	2	09/22/2019	Google Scholar
HART protocols for authentication	16,600 results	0	09/22/2019	Google Scholar
HART protocol authentication	10,900 results	0	09/22/2019	Google Scholar
HART protocol authentication in smart home networks	4,760 results	2	09/22/2019	Google Scholar
HART protocol authentication with IOT devices	13,200 results	1	09/22/2019	Google Scholar
HART protocol design	477,000 results	0	09/22/2019	Google Scholar
HART protocol	1,020,000 results	0	09/22/2019	Google Scholar
HART protocol comparison	637,000 results	1	09/22/2019	Google Scholar
HART protocol comparison for authentication	13,400 results	0	09/22/2019	Google Scholar
HART protocol standard	585,000 results	2	09/22/2019	Google Scholar
Domain Controller for Smart Home	148,000 results	2	9/28/2019	Google Scholar
Domain controller for iot authentication	15,100 results	5	9/28/2019	Google Scholar
Domain controller for iot	28,900 results	0	09/28/2019	Google Scholar
Domain Controller Network	566,000 results	1	09/28/2019	Google Scholar
Domain Controller Network architecture	174,000 results	0	9/28/2019	Google Scholar
Domain Controller Network architecture for smart home	40,600 results	0	9/28/2019	Google Scholar
authentication Domain Controller	36,600 results	6	09/28/2019	Google Scholar
authentication Domain Controller smart home	36,600 results	1	09/28/2019	Google Scholar
smart home authentication	84,700 results	2 2	09/28/2019	Google Scholar
smart home identification	1,750,000 results	0	09/28/2019	Google Scholar
smart home device authentication	54,400 results	4	09/28/2019	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
smart home device authentication method	45,100 result	0	09/28/2019	Google Scholar
Smart Home Platforms	272,000 results	0	09/28/2019	Google Scholar
smart home platform authentication	48,300 results	0	09/28/2019	Google Scholar
smart grid device authentication	54,400 results	1	09/28/2019	Google Scholar
smart grid device authentication method	45,100 results	0	09/28/2019	Google Scholar
smart homes security technologies	341,000 results	0	09/28/2019	Google Scholar
smart homes authentication technologies	32,900 results	0	09/28/2019	Google Scholar
VPN technology for smart homes	15,300 results	0	09/28/2019	Google Scholar
VPN for smart homes	15,600 results	0	09/28/2019	Google Scholar
VPN for smart home authentication	8,770 results	0	09/28/2019	Google Scholar
authentication protocol verification methods comparison	307,00	5	03/04/2020	Google Scholar
protocol verification methods	2,770,000	4	03/04/2020	Google Scholar
protocol verification	2,670,000	4	03/04/2020	Google Scholar
smart metering communication protocol	129,00	2	03/04/2020	Google Scholar
powerline communication protocol	36,600	3	03/04/2020	Google Scholar
Evaluation method for smart home and smart grid authentication protocols	29,100	1	03/04/2020	Google Scholar
protocol evaluation	4,740,000	1	03/04/2020	Google Scholar
protocol evaluation method	5,640,000	1	03/04/2020	Google Scholar
authentication protocol evaluation	477,00	2	03/04/2020	Google Scholar
authentication protocol evaluation method	514,000	1	03/04/2020	Google Scholar
smart home protocol evaluation	564,000	2	03/04/2020	Google Scholar
smart home protocol evaluation method	422,000	1	03/04/2020	Google Scholar
smart grid protocol evaluation	203,000	2	03/04/2020	Google Scholar
smart grid protocol evaluation method	170,00	1	03/04/2020	Google Scholar
EAP Authentication	24,600	5	03/04/2020	Google Scholar
EAP TLS	11,800	3	03/04/2020	Google Scholar

Continued on next page

Table 12 – *Continued from previous page*

Term searched	Total results	Used works	Date	Source
MSCHAPv2	1,310	2	03/04/2020	Google Scholar
EAP TTLS	3,410	6	03/04/2020	Google Scholar
EAP	482,000	4	03/04/2020	Google Scholar
EAPPSK	374	1	03/04/2020	Google Scholar
EAPTLS	6,150	2	03/04/2020	Google Scholar
EAPTTLS	2,790	1	03/04/2020	Google Scholar
PEAP-MSCHAPv2	181	1	03/04/2020	Google Scholar
ProVerif	2,700	12	03/04/2020	Google Scholar
BAN logic	675,000	14	03/04/2020	Google Scholar

Table 12: Searched Terms

References

[1] 3GPP, "Formal analysis of 3g authentication and key agreement protocol," Tech. Rep., 2001.

[2] M.-S. Kim and S. Valcourt, "Selecting a standard outer method for eap," 01 2006.

[3] K. M. J. Haataja, "Security in bluetooth, wlan and irda: a comparison," Department of Computer Science, Kuopio, Finland, Tech. Rep., 2006.

[4] E. A. Inc., "Security of enocean radio networks v2.3," Tech. Rep., 2018.

[5] K. A. Alezabi, F. Hashim, S. J. Hashim, B. M. Ali, and A. Jamalipour, "Authentication process enhancements in wimax networks," *Sec. and Commun. Netw.*, vol. 9, no. 17, p. 4703–4725, 2016.

[6] J. NOBRIOT and B. DEWASSIE, "Z-wave network-protocol command class specification," Tech. Rep., 2020.

[7] T. Group, "Thread commissioning," Tech. Rep., 2015.

[8] IBM, "An overview of the ssl or tls handshake," Available at https://www.ibm.com/support/knowledgecenter/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm#sy10660___sy10660_2 Accessed : 2019/11/12.

[9] Utsav, Chiraag, Samuel, and Anantha, "eedtls: Energy-efficient datagram transport layer security for the internet of things," 12 2017, pp. 1–6.

[10] R. Borgaonkar, "Authentication and related threats in 2g/3g/4g networks," Available at https://coinsrs.no/wp-content/uploads/2016/08/metochi2016-Borgaonkar-authentication-in-2g3g4g-networks.pdf Accessed : 2020/6/5.

[11] Wikipedia, "Cbc-mac," Available at https://en.wikipedia.org/wiki/CBC-MAC Accessed : 2020/11/12, 2019.

[12] Intel, "Intel," 03 2019. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000005725/network-and-i-o/wireless-networking.htmAccessed2019/11/10

[13] C. Rensing, M. Karsten, and B. Stiller, "A survey on aaa mechanisms, protocols, and architectures and a policy-based approach beyond: Ax," https://doi.org/10.3929/ethz-a-004283995, Department of Computer Science, Zurich, Switzerland, Tech. Rep., 2001.

[14] B. Lloyd and W. Simpson, "Ppp authentication protocols," https://tools.ietf.org/html/rfc1334 Accessed : 2020/04/07, October 1992.

[15] L. Blunk and J. Vollbrecht, "Ppp extensible authentication protocol (eap)," https://tools.ietf.org/html/rfc1334 Accessed : 2020/04/07, March 1998.

[16] I. Ali, S. Sabir, and Z. Ullah, "Internet of things security, device authentication and access control: A review," *International Journal of Computer Science and Information Security*, vol. 14, no. 8, pp. 457–466, 2016.

[17] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Transactions on Computer Systems*, vol. 8, no. 1, pp. 18–36, 1990.

[18] Y. Ismail, "Internet of things (iot) for automated and smart applications," IntechOpen, 2019.

[19] J. R. Ltd, "Juniper research," Available at https://www.juniperresearch.com/resources/infographics/smart-home-statistics Accessed : 2019/11/12, November 2009.

[20] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, "Smart grid technologies: Communication technologies and standards," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 529–539, 2011.

[21] T. Wang, "Statista," Available at https://www.juniperresearch.com/resources/infographics/smart-home-statistics Accessed : 2019/11/12, June 2019.

[22] L. Jianchen, Z. Jianguang, F. Jingjing, and D. Juxing, "The security ecurity ecurity ecurity research research research research of network network network network access control control control control system," First ACIS International Symposium on Cryptography, and Network Security, Data Mining and Knowledge Discovery, E-Commerce and Its Applications, and Embedded Systems, 2010.

[23] M. Ryan, M.Talabis, R. McPherson, I. Miyamoto, J. L. Martin, and D.Kaye, *Access Analytics*, ser. Information Security Analytics. Syngress, 2015.

[24] A. Baviskar, J. Baviskar, A. M. S. Wagh, and P. Dave, "Comparative study of communication technologies for power optimized automation systems: A review and implementation," Fifth International Conference on Communication Systems and Network Technologies, 2015.

[25] E. Ferro and F. Potorti, "Bluetooth and wi-fi wireless protocols: a survey and a comparison," *IEEE Wireless Communications*, vol. 12, no. 1, pp. 12–26, 2005.

[26] T. Lennvall, S. Svensson, and F. Hekland, "A comparison of wirelesshart and zigbee for industrial applications," IEEE International Workshop on Factory Communication Systems, 2008.

[27] A. J. D. Rathnayaka, V. M. Potdar, and S. J. Kuruppu, "Evaluation of wireless home automation technologies," 5th IEEE International Conference on Digital Ecosystems and Technologies, 2011.

[28] C. Withanage, R. Ashok, C. Yuen, and K. Otto, "A comparison of the popular home automation technologies," IEEE Innovative Smart Grid Technologies, 2014.

[29] M. B. Tamboli and D. Dambawade, "Secure and efficient coap based authentication and access control for internet of things (iot)," IEEE International Conference On Recent Trends In Electronics Information Communication Technology, 2016.

[30] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, "A review of smart homes—past, present, and future," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1190–1203, 2012.

[31] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, "Smart grid technologies: Communication technologies and standards," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 529–539, November 2011.

[32] L. Wenpeng, D. Sharp, and S. Lancashire, "Smart grid communication network capacity planning for power utilities," IEEE PES T&D 2010, Transmission Distrib. Conf. Expo., New Orleans, April 2010.

[33] V. C. Gungor, D. Sahin, T. Kocak, and S. Ergut, "Smart grid communications and networking," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 529–539, 2011.

[34] B. Sidhu, H. Singh, and A. Chhabra, “Emerging wireless standards: Wifi, zigbee and wimax,” *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, vol. 1, no. 1, pp. 43–48, 2007.

[35] P. P. Gaikwad, J. P. Gabhane, and S. S. Golait, “3-level secure kerberos authentication for smart home systems using iot,” 1st International Conference on Next Generation Computing Technologies, 2015.

[36] R. Maheshwari, A. Gupta, and N. Chandra, “Secure authentication using biometric templates in kerberos,” 2nd International Conference on Computing for Sustainable Global Development, 2015.

[37] P. Barker and A. Boucouvalas, “Performance modeling of the irda protocol for infrared wireless communications,” *IEEE Communications Magazine*, vol. 36, no. 12, pp. 113–117, 1998.

[38] Samaras, O’Brian, and Edwards, “Indoor optical wireless systems - a review,” in *Optical and Quantum Electronics*, 1997.

[39] A. T. Lodamo, “M2m protocols, solutions and platforms for smart home environments,” Master’s thesis, MID SWEDEN UNIVERSITY, Sundsvall, 2012.

[40] A. I. Gardezi, “Security in wireless cellular networks,” https://www.cse.wustl.edu/~jain/cse574-06/ftp/cellular_security/index.html Accessed : 2020/04/07, April 2006.

[41] A. N. Nokia, J. Arkko, and V. Torvinen, “Hypertext transfer protocol (digest authentication using authentication and key agreement (aka),” https://tools.ietf.org/html/rfc3310. Accessed 2019/10/11, September 2002.

[42] H. Haverinen and J. Salowey, “Extensible authentication protocol method for global system for mobile communications (gsm) subscriber identity modules (eap-sim),” https://tools.ietf.org/html/rfc4186 Accessed : 2020/05/03, January 2006.

[43] K. Prakasha, “Authentication and key agreement in 3gpp networks,” vol. 5, 07 2015, pp. 143–154.

[44] F. S. Inc, “Long term evolution protocol overview,” Tech. Rep., 2008.

[45] H.-J. Seo and H.-W. Kim, “Network and data link layer security for dash7,” *Journal of information and communication convergence engineering*, vol. 10, 09 2012.

[46] M. Weyn, G. Ergeerts, L. Wante, C. Vercauteren, and P. Hellinckx, "Survey of the dash7 alliance protocol for 433 mhz wireless sensor communication," *International Journal of Distributed Sensor Networks*, vol. 2013, p. 9, 10 2013.

[47] U. Mehboob, Q. Zaib, and C. Usama, *Survey of IoT Communication Protocols Techniques, Applications, and Issues.* xFlow Research Inc, 2016.

[48] A. Chirumamilla, H. Seo, D. Lee, and H. Kim, "Implementation of an rfid key management system for dash7," *Journal of information and communication convergence engineering*, vol. 12, 03 2014.

[49] M. Weyn, G. Ergeerts, R. Berkvens, B. Wojciechowski, and Y. Tabakov, "Dash7 alliance protocol 1.0: Low-power, mid-range sensor and actuator communication," in *IEE Conferenece on Standards for Communications and Networking*, Tokyo, 2015.

[50] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi," in *The 33rd Annual Conference of the IEEE Industrial Electronics Society*, Taipei, 2007.

[51] J. T. Vainio, "Bluetooth security," http://www.yuuhaw.com/bluesec.pdf Accessed : 2020/04/07, 2005.

[52] EnOcean, "Enocean radio protocol 2," Tech. Rep., 2017.

[53] G. C. Inc, "Network keys and the ant+ managed network," https://www.thisisant.com/developer/resources/tech-bulletin/network-keys-and-the-ant-managed-network Accessed : 2020/04/07, 2013.

[54] L. Camelo, A. Greene, J. Loving, and U. Otgonbaatar, "The internet of insecure things analyzing a low-energy protocol and cryptographic solutions," Master's thesis, MIT, 2015.

[55] Y. Peizhong, A. Iwayemi, and C. Zhou, "Developing zigbee deployment guideline under wifi interference for smart grid applications," *IEEE Transactions on Smart Grid*, vol. 2, no. 1, pp. 110–120, March 2011.

[56] H. P. A. I., "Smart energy profile 2 application protocol standard, zigbee alliance," *ZigBee Alliance Inc*, 2013.

[57] G. Thonet, P. Allard-Jacquin, and P. Colle, *ZigBee – WiFi Coexistence.* Grenoble: Schneider Electric, 2008.

[58] H. R. Nielson, E. Yuksel, and F. Nielson, "Zigbee-2007 security essentials," in *Proceedings of The 13. Nordic Workshop on Secure IT Systems*, 2008.

[59] I. Unwala and J. Lu, "Iot protocols : Z-wave and thread," *International Journal on Future Revolution in Computer Science & Communication Engineering*, vol. 3, no. 11, pp. 355–359, 2017.

[60] W. Rzepecki, L. Iwanecki, and P. Ryba, "Ieee 802.15.4 thread mesh network – data transmission in harsh environment," in *6th International Conference on Future Internet of Things and Cloud Workshops*, 2018.

[61] M. B. Yassein, W. Mardini, and A. Khalil, "Smart homes automation using z-wave protocol," in *International Conference on Engineering & MIS*, 2016.

[62] I. Unwala, Z. Taqvi, and J. Lu, "Iot security : Zwave and thread," in *IEEE Green Technologies Conference*, Austin, 2018.

[63] JFR, NOBRIOT, BBR, and DEWASSIE, "Z-wave transport-encapsulation command class specification," Tech. Rep., 2020.

[64] M. Caneill and J.-L. Gilis, "Attacks against the wifi protocols wep and wpa," https://matthieu.io/dl/papers/wifi-attacks-wep-wpa.pdf Accessed : 2020/04/07, 2010.

[65] PureVPN, "Wifi security protocols- difference between wep and wpa," https://www.purevpn.com/wifi-vpn/security-protocols Accessed : 2020/04/07.

[66] Cyberpunk, "Wireless security protocols: Wep, wpa, wpa2 and wpa3," https://www.cyberpunk.rs/wireless-security-protocols-wep-wpa-wpa2-and-wpa3 Accessed : 2020/04/07.

[67] SecureW2, "Simplifying wpa2-enterprise and 802.1x," https://www.securew2.com/solutions/wpa2-enterprise-and-802-1x-simplified/ Accessed : 2020/04/07.

[68] W.-F. Alliance, "Wpa3 specification version 2," Tech. Rep., 2019.

[69] ——, "Wpa3 security considerations," Tech. Rep., 2019.

[70] E. International, "High rate ultra wideband phy and mac standard," Tech. Rep., 2008.

[71] Q. Guan and Q. Guan, "Analysis of security mechanism in uwb standard of ecma368," 01 2007.

[72] M. S. I. M. Zin and M. Hope, "A review of uwb mac protocols," 06 2010, pp. 526 – 534.

[73] T. Haider, "Wireless communication using wimax technology," vol. 14, pp. 142–161, 09 2010.

[74] U. Kucharzewski and Z. Kotulski, "Wimax networks-architecture and data security," *Annales UMCS, Informatica*, vol. 10, pp. 177–185, 01 2010.

[75] S. Xu and C.-T. Huang, "Attacks on pkm protocols of ieee 802.16 and its later versions," 10 2006, pp. 185 – 189.

[76] T. Gonnot and J. Saniie, "User defined interactions between devices on a 6lowpan network for home automation," in *IEEE International Technology Management Conference*, Chicago, 2014.

[77] A. Badach, "Protocol structure of 6lowpan devices," Tech. Rep., 09 2017.

[78] J. Olsson, "6lowpan demystified," Tech. Rep., 2014.

[79] D. M. Laverty, D. J. Morrow, R. Best, and P. A. Crossley, "Telecommunications for smart grid: Backhaul solutions for the distribution network," in *IEEE PES General Meeting*, Providence, 2010.

[80] V. Oksman and S. Galli, "G.hn: The new itu-t home networking standard," *IEEE Communications Magazine*, vol. 47, no. 10, pp. 138–145, 2009.

[81] ITU-T, "Password-authenticated key exchange (pak) protocol," Tech. Rep., 2007.

[82] F. Division, "G.hn - next generation home network technology," Tech. Rep.

[83] E. Corporation, *LonTalk Protocol Specification.* Palo Alto: Echelon Corp, 1994.

[84] ——, "Introduction to the lonworks platform revision 2," Tech. Rep., 2009.

[85] R. P. Lewis, P. Igict, and Z. Zhou, "Assessment of communication methods for smart electricity metering in the u.k," in *Sustainable Alternative Energy (SAE), IEEE PES/IAS*, Valencia, 2009.

[86] J. Luansheng, L. Chunxia, G. Xiumei, and M. Chongxiao, "The design of intelligent lighting system in college classroom," in *International Conference on Future Electrical Power and Energy Systems*, Sanya, 2012.

[87] T. H. C. Assistant, "Appendix 8 universal powerline bus (upb)," Tech. Rep.

[88] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and E. H. Levkowetz, "Extensible authentication protocol (eap)," https://www.ietf.org/rfc/rfc3748.txt Accessed : 2020/04/07.

[89] J.-C. Chen and Y.-P. Wang, "Extensible authentication protocol (eap) and ieee 802.1x: tutorial and empirical experience," *IEEE Communications Magazine*, vol. 43, no. 12, 2005.

[90] I. Simply Automated, "The upb system description," April 2005.

[91] K. Association, "Knx system arguments," http://knx.com.ua/attachments/article/132/KNX-basic_course_full.pdf Accessed : 2020/04/08, Tech. Rep.

[92] S. Cavalieri and G. Cutuli, "Implementing encryption and authentication in knx using diffie-hellman and aes algorithms," in *35th Annual Conference of IEEE Industrial Electronics*, Porto, 2009.

[93] wanderingsamurai.net, "Interface communication protocol," Available at https://wanderingsamurai.net/electronics/cm11a-x10-protocol-document Accessed : 2019/11/12.

[94] smartlabs, "Insteon developer's guide," Tech. Rep., 2007.

[95] G. Odinak, "Automated home control using existing electrical lines as a communications medium," U.S. Patent 5 929 748A, 12 17, 1998.

[96] G. Bakshi and A. Dearien, "Back to the basics: what is hart protocol and how does it work?" *Texas Instruments*, 01 2018. [Online]. Available: https://e2e.ti.com/blogs_/b/analogwire/archive/2018/01/26/back-to-the-basics-what-is-hart-protocol-and-how-does-it-workAccessed09/11/2019

[97] S. Raza, A. Slabbert, T. Voigt, and K. Landernas, “Security considerations for the wirelesshart protocol,” 10 2009, pp. 1 – 8.

[98] Emerson, “System engineering guidelines iec 62591 wirelesshart,” Tech. Rep., 2016.

[99] B. Blanchet, V. Cheval, X. Allamigeon, B. Smyth, and M. Sylvestre, “Proverif: Cryptographic protocol verifier in the formal model,” Available at https://prosecco.gforge.inria.fr/personal/bblanche/proverif/ Accessed : 2019/12/11.

[100] M. Beadles and D. Mitton, “Criteria for evaluating network access server protocols,” RFC 3169 https://tools.ietf.org/html/rfc3169 Accessed : 2020/01/12, September 2001.

[101] E. G. Jones, “Operational security requirements for large internet service provider (isp) ip network infrastructure,” Available at http://www.hjp.at/doc/rfc/rfc3871.html Accessed : 2020/01/12, September 2004.

[102] E. J. Martocci, P. D. Mil, N. Riou, and W. Vermeylen, “Building automation routing requirements in low-power and lossy networks,” RFC 5867 http://www.hjp.at/doc/rfc/rfc5867.html Accessed : 2020/01/12, June 2010.

[103] E. Stokes, D. Byrne, B. Blakley, and P. Behera, “hjp: doc: Rfc 2820: Access control requirements for ldap,” RFC 2820 http://www.hjp.at/doc/rfc/rfc2820.html Accessed : 2020/01/12, May 2000.

[104] Z. Shelby, K. Hartke, and C. Bormann, “The constrained application protocol (coap),” RFC 7252 http://www.hjp.at/doc/rfc/rfc7252.html Accessed : 2020/01/12, June 2014.

[105] E. Stokes, R. Weiser, R. Moats, and R. Huber, “Lightweight directory access protocol (version 3) replication requirements,” RFC 3384 http://www.hjp.at/doc/rfc/rfc3384.html Accessed : 2020/01/12, October 2002.

[106] E. Rescorla, “The transport layer security (tls) protocol version 1.3,” RFC 8446 https://tools.ietf.org/html/rfc8446 Accessed : 2020/01/12, August 2018.

[107] N. Brownlee, “Accounting requirements for ipng,” RFC 1672 http://www.hjp.at/doc/rfc/rfc1672.html Accessed : 2020/01/12, August 1994.

[108] M. Parthasarathy, "Protocol for carrying authentication and network access (pana) threat analysis and security requirements," RFC 4016 http://www.hjp.at/doc/rfc/rfc4016.html Accessed : 2020/01/12, March 2005.

[109] F. O. for Information Security, "Cryptographic mechanisms: Recommendations and key lengths," Tech. Rep., 2020.

[110] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "Proverif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial," Tech. Rep., 2018.

[111] J. Hernandez-Castro, A. Alcaide, and J. Torres, "Validating the use of ban logic," in *Computational Science and Its Applications - ICCSA*, Assisi, Italy, 2004.

[112] A. Saabas and T. Uustalu, "A compositional natural semantics and hoare logic for low-level languagesn," *Electron. Notes Theor. Comput. Sci*, vol. 156, no. 1, p. 151–168, 2006.

[113] TutorialsPoint, "Gsm - specification," https://www.tutorialspoint.com/gsm/gsm_specification.htm Accessed : 2020/04/08.

[114] E. Jorg, H.-J. V. C. Bettstetter, and C. Hartmann, *GSM - Architecture, Protocols and Services (3. ed.).*, 01 2009.

[115] ETSI, "Universal mobile telecommunications system (umts), medium access control (mac) protocol specification (3gpp ts 25.321 version 8.16.0 release 8)," Tech. Rep., 2012.

[116] H. Kaaranen., A. Ahtiainen, L. Laitinen, S.Naghian, and V. Niemi, *UMTS Networks: Architecture, Mobility and Services: Second Edition*, 11 2005.

[117] ETSI, "Etsi ts 128 302 v14.0.0," Tech. Rep., 2017.

[118] ——, "Digital cellular telecommunications system (phase 2), mobile application part (map) specification (gsm 09.02)," Tech. Rep., 1996.

[119] ——, "Etsi ts 132 773 v9.0.0," Tech. Rep., 2010.

[120] M. Abdeljebbar and R. Kouch, "Security improvements of eps-aka protocol," *International Journal of Network Security*, vol. 20, p. 636, 09 2017.

[121] W. Ayoub, A. E. Samhat, F. Nouvel, M. Mroue, and J. Prevotet, "Internet of mobile things: Overview of lorawan, dash7, and nb-iot in lpwans standards and supported mobility," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1561–1581, 2019.

[122] T. A. Yahiya, *Understanding LTE and its Performance*, 05 2011, pp. 55–73.

[123] TutorialsPoint, "Gsm - security and encryption," https://www.tutorialspoint.com/gsm/gsm_security.htm Accessed : 2020/04/08.

[124] irelandscape, "Introduction to mobile networks - 3g (umts) authentication," https://steemit.com/mobilenetworks/@irelandscape/introduction-to-mobile-networks-3g-umts-authentication Accessed : 2020/04/07, 2018.

[125] S. Alt, P.-A. Fouque, G. Macario-Rat, C. Onete, and B. Richard, "A cryptographic analysis of umts/lte aka," 06 2016, pp. 18–35.

[126] ETSI, "Using cellular algorithms," https://www.etsi.org/security-algorithms-and-codes/cellular-algorithm-licences Accessed : 2020/04/08.

[127] K. Kumar, G. Shailaja, K. Ammayappan, and A. Saxena, "Mutual authentication and key agreement for gsm," 07 2006, pp. 25 – 25.

[128] ETSI, "Digital cellular telecommunications system (phase 2), universal mobile telecommunications system (umts), direct tunnel deployment guideline (3gpp tr 23.919 version 7.0.0 release 7)," Tech. Rep., 2007.

[129] A. Technology, "Gprs tunneling protocol (gtp) processing," Tech. Rep.

[130] I. Unwala, Z. Taqvi, and J. Lu, "Thread: An iot protocol," in *2018 IEEE Green Technologies Conference (GreenTech)*, 2018, pp. 161–167.

[131] E. Corporation, *LonTalk Protocol Specification, LonWorks Engineering Bulltein.* Palo Alto: Echelon Corp, 1993.

[132] L. M. L. Oliveira, J. J. P. C. Rodrigues, A. F. de Sousa, and J. Lloret, "A network access control framework for 6lowpan networks," Tech. Rep., 2013.

[133] KNX, "Knx basics," Available at http://knx.fi/doc/esitteet/KNX-Basics_en.pdf Accessed : 2019/11/12, Tech. Rep.

[134] V. Lourdas, "Knx ip secure," Available at https://support.knx.org/hc/en-us/articles/360012666599-KNX-IP-Secure Accessed : 2019/11/12.

[135] KNX, "Knx security position paper," http://knx.fi/doc/esitteet/KNX-Security-Position-Paper_en.pdf Accessed : 2020/04/07, Tech. Rep.

[136] ABR, "Introduction to the z-wave security ecosystem," Tech. Rep., 2016.

[137] ThisIsAnt, "Ant message protocol and usage," Tech. Rep., 2014.

[138] C. Badenhop, S. Graham, B. Ramsey, B. Mullins, and L. Mailloux, "The z-wave routing protocol and its security implications," *Computers & Security*, vol. 68, 04 2017.

[139] C. Hager and S. Midkiff, "An analysis of bluetooth security vulnerabilities," vol. 3, 04 2003, pp. 1825 – 1831 vol.3.

[140] M. Conti and D. Moretti, "System level analysis of the bluetooth standard," 03 2005, pp. 118–123.

[141] J. Sanchez-Gomez, D. Garcia-Carrillo, R. Marin-Perez, and A. F. Skarmeta, "Secure authentication and credential establishment in narrowband iot and 5g," Tech. Rep., 2020.

[142] ITU-T, "Series y: Global information infrastructure, internet protocol aspects and next-generation networks, internet of things and smart cities," Tech. Rep., 2016.

[143] M. Weyn, G. Ergeerts, R. Berkvens, B. Wojciechowski, and Y. Tabakov, "Dash7 alliance protocol 1.0 low-power, mid-range sensor and actuator communication," 10 2015.

[144] 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)*, 2014, pp. 1–10.

[145] E. F. Hao, "J-pake: Password authenticated key exchange by juggling draft-hao-jpake-01," https://tools.ietf.org/html/draft-hao-jpake-01 Accessed : 2020/04/07, December 2015.

[146] ——, "Schnorr nizk proof: Non-interactive zero knowledge proof for discrete logarithm draft-hao-schnorr-01," https://tools.ietf.org/html/rfc5021 Accessed : 2020/01/12, December 2015.

[147] Microsoft, "Ms-chap: Extensible authentication protocol method for microsoft challenge handshake authentication protocol (chap)," https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-chap/4740bf05-db7e-4542-998f-5a4478768438 Accessed : 2020/04/07.

[148] F. Bersani, "The eap-psk protocol: A pre-shared key extensible authentication protocol (eap) method," Available at https://tools.ietf.org/html/draft-bersani-eap-psk-00 Accessed : 2019/11/12, 2004.

[149] J.-K. Tsay and S. Mjolsnes, "A vulnerability in the umts and lte authentication and key agreement protocols," 10 2012, pp. 65–76.

[150] C. . H. C. for Information Security, "Cispa - helmholtz center for information security," Available at https://people.cispa.io/cas.cremers/scyther/ Accessed : 2019/12/11, April 2014.

[151] C. C. J. D. S. M. R. S. B. S. D. Basin, "Tamarin prover," Available at https://tamarin-prover.github.io/ Accessed : 2019/11/12.

[152] P. W. Security, "Portswigger ltd," Available at https://portswigger.net/burp Accessed : 2019/12/11.

[153] T. Lennvall, S. Svensson, and F. Hekland, "A comparison of wirelesshart and zigbee for industrial applications," in *Factory Communication Systems, 2008. WFCS 2008. IEEE International Workshop*, Dresden, 2008.

[154] S. E. International, "Smart energy international," *Smart Energy International*, 09 2003. [Online]. Available: https://www.smart-energy.com/regional-news/europe-uk/the-euridis-protocol-an-open-solution-for-amr-using-various-media/ Accessed10/11/2019

[155] A. Fox and S. D. Gribble, "Security on the move: Indirect authentication using kerberos," in *Proceedings of the 2nd annual international conference on Mobile computing and networking*, 1996.

[156] E. El-Emam, M. Koutb, H. Kelash, and O. S. Faragallah, "An authentication protocol based on kerberos 5," *International Journal of Network Security*, vol. 12, no. 2, pp. 147–158, 2011.

[157] G. Bella and E. Riccobene, “Formal analysis of the kerberos authentication system,” *Journal of Universal Computer Science*, vol. 3, no. 12, pp. 1337–1381, 1997.

[158] G. Bella and L. Paulson, “Kerberos version iv: Inductive analysis of the secrecy goals,” in *Computer Security - ESORICS 98.* Springer, 1998, pp. 361–375.

[159] S. Bellovin and M. Merrit, “Limitations of the kerberos authentication system,” *SIGCOMM Computer Communication Review*, vol. 20, no. 5, pp. 119–132, 1990.

[160] W. Stallings, “Cryptography and network security principles and practices,” *Upper Saddle River: Pearson Prentice Hall*, 2006.

[161] S. Cavalieri, G. Cutuli, and M. Malgeri, “A study on security mechanisms in knx-based home/building automation,” in *IEEE 15th Conference on Emerging Technologies & Factory Automation*, Bilbao, 2010.

[162] G. Bovet and J. Hennebert, “Web-of-things gateway for knx networks,” Erlangen/Nuremberg, 2013.

[163] H.-J. Langels, “Knx ip – using ip networks as knx medium,” in *KNX Scientific Conference*, Porto, 2008.

[164] J. A. Nazabal, F. Falcone, C. Fernandez-Valdivielso, and I. R. Matias, “Proposal for improving connectivity and adding authentication and security to knxnet/ip protocol,” *International Journal of Smart Home*, vol. 8, no. 2, pp. 77–90, 2014.

[165] J. A. Nazabal, F. Falcone, S. C. Mukhopadhyay, and I. R. Matias, “Accessing knx devices using usb/knx interfaces for remote monitoring and storing sensor data,” *International Journal of Smart Home*, vol. 7, no. 2, pp. 105–110, 2013.

[166] J. G. Steiner, C. Neuman, and J. I. Schiller, “Kerberos: an authentication service for open network systems,” in *USENIX Winter Conference*, Dallas, 1988.

[167] M. Bungart, C. Fohry, and J. Posner, “Fault-tolerant global load balancing in x10,” in *16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, 2014.

[168] D. Whiting, R. Housley, V. Security, and N. Ferguson, “Counter with cbc-mac (ccm),” RFC 3610 https://tools.ietf.org/html/rfc3610 Accessed 11/11/2019, September 2003.

[169] M. Shabanzadeh and M. P. Moghaddam, “What is the smart grid? definitions, perspectives, and ultimate goals,” in *International Power System Conference (PSC)*, Tehran, 2013.

[170] S. Josefsson, “Using kerberos version 5 over the transport layer security (tls) protocol,” RFC 6251 https://tools.ietf.org/html/rfc6251 Accessed : 2020/01/12, May 2011.

[171] T. Dierks and E. Rescorla, “The transport layer security (tls) protocol version 1.1,” RFC 4346 https://tools.ietf.org/html/rfc4346 Accessed : 2020/01/12, April 2006.

[172] ——, “The transport layer security (tls) protocol version 1.2,” RFC 5246 https://tools.ietf.org/html/rfc5246 Accessed : 2020/01/12, August 2008.

[173] D. Mitton, “Network access servers requirements: Extended radius practices,” RFC 2882 http://www.hjp.at/doc/rfc/rfc2882.html Accessed : 2020/01/12, July 2000.

[174] J. Galvin and K. McCloghrie, “Security protocols for version 2 of the simple network management protocol (snmpv2),” RFC 1446 http://www.hjp.at/doc/rfc/rfc1446.html Accessed : 2020/01/12, April 1993.

[175] S. Farrell, J. Vollbrecht, P. Calhoun, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence, “Aaa authorization requirements,” RFC 2906 http://www.hjp.at/doc/rfc/rfc2906.html Accessed : 2020/01/12, August 2000.

[176] J. Loughney and G. Camarillo, “Authentication, authorization, and accounting requirements for the session initiation protocol (sip),” RFC 3702 http://www.hjp.at/doc/rfc/rfc3702.html Accessed : 2020/01/12, February 2004.

[177] E. M. Brunner, “Requirements for signaling protocols,” RFC 3726 http://www.hjp.at/doc/rfc/rfc3726.html Accessed : 2020/01/12, April 2004.

[178] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “Sip: Session initiation protocol,” RFC 3261 http://www.hjp.at/doc/rfc/rfc3261.html Accessed : 2020/01/12, June 2002.

[179] S. Kelly and S. Ramamoorthi, “Requirements for ipsec remote access scenarios,” RFC 3457 http://www.hjp.at/doc/rfc/rfc3457.html Accessed : 2020/01/12, January 2003.

[180] J. Schiller, “Strong security requirements for internet engineering task force standard protocols,” RFC 3365 http://www.hjp.at/doc/rfc/rfc3365.html Accessed : 2020/01/12, August 2002.

[181] K. Raeburn, “Encryption and checksum specifications for kerberos 5,” RFC 3961 https://tools.ietf.org/html/rfc3961 Accessed : 2020/01/12, February 2005.

[182] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, “The kerberos network authentication service (v5),” RFC 4120 https://tools.ietf.org/html/rfc4120 Accessed : 2020/01/12, July 2005.

[183] S. Hartman and L. Zhu, “A generalized framework for kerberos pre-authentication,” RFC 6113 https://tools.ietf.org/html/rfc6113 Accessed : 2020/01/12, April 2011.

[184] L. H. Astrand and T. Yu, “Deprecate des, rc4-hmac-exp, and other weak cryptographic algorithms in kerberos,” RFC 6649 https://tools.ietf.org/html/rfc6649 Accessed : 2020/01/12, July 2012.

[185] E. S. Hartman, K. Raeburn, and L. Zhu, “Kerberos principal name canonicalization and cross-realm referrals,” RFC 6806 https://tools.ietf.org/html/rfc6806 Accessed : 2020/01/12, November 2012.

[186] S. Sorce and T. Yu, “Kerberos authorization data container authenticated by multiple message authentication codes (macs),” RFC 7751 https://tools.ietf.org/html/rfc7751 Accessed : 2020/01/12, March 2016.

[187] L. Zhu, P. Leach, S. Hartman, and E. S. Emery, “Anonymity support for kerberos,” RFC 8062 https://tools.ietf.org/html/rfc8062 Accessed : 2020/01/12, February 2017.

[188] A. Jainand, N. Kinder, and N. McCallum, “Authentication indicator in kerberos tickets,” RFC 8129 https://tools.ietf.org/html/rfc8129 Accessed : 2020/01/12, March 2017.

[189] B. Kaduk and M. Short, “Deprecate triple-des (3des) and rc4 in kerberos,” RFC 8429 https://tools.ietf.org/html/rfc8429 Accessed : 2020/01/12, October 2018.

[190] L. Zhu, P. Leach, and K. Jaganathan, “Kerberos cryptosystem negotiation extension,” RFC 4537 https://tools.ietf.org/html/rfc4537 Accessed : 2020/01/12, June 2006.

[191] L. Hornquist and S. Hartman, “Generic security service application program interface (gss-api): Delegate if approved by policy,” RFC 5896 https://tools.ietf.org/html/rfc5896 Accessed : 2020/01/12, June 2010.

[192] S. Josefsson, “Extended kerberos version 5 key distribution center (kdc) exchanges over tcp,” RFC 5021 https://tools.ietf.org/html/rfc5021 Accessed : 2020/01/12, August 2007.

[193] Q. Li, F. Yang, H. Zhu, and L. Zhu, “Formal modeling and analyzing kerberos protocol,” in *World Congress on Computer Science and Information Engineering*, Bilbao, 2009.

[194] I. U. the Artificial Intelligence Laboratory (AI-Lab) at DIST, Università di Genova, F. I. the CASSIS group at INRIA, Nancy, S. E. the Information Security Group at ETHZ, Zurich, and G. S. Siemens AG, Munich, “Automated validation of internet security protocols and applications,” Available at http://www.avispa-project.org/ Accessed : 2020/04/01, June 2006.

[195] M. M. andBorka Jerman Blažič and S. Josimovski, “Quantifying usability and security in authentication,” in *35th IEEE Annual Computer Software and Applications Conference*, 2011.

[196] H. Glanzer, L. Krammer, and W. Kastner, “Increasing security and availability in knx networks,” in *Sicherheit*, 2016.

[197] smarthome.com, “What is x10?” https://www.smarthome.com/sc-what-is-x10-home-automation Accessed : 2020/04/07.

[198] D. Lechner, W. Granzer, and W. Kastner, “Security for knxnet/ip,” Tech. Rep., 2008.

[199] TutorialsPoint, “Gsm - protocol stack,” https://www.tutorialspoint.com/gsm/gsm_protocol_stack.htm Accessed : 2020/04/08.

[200] R. Pinheiro, A. Aguiar, P. Pinheiro, A. Neto, R. Cunha, and D. Neto, “Scalability analysis of a model for gsm mobile network design,” 01 2008, pp. 465–469.

[201] Y. Beyene, R. Jantti, K. Ruttik, and S. Iraji, “On the performance of narrow-band internet of things (nb-iot),” 03 2017, pp. 1–6.

[202] V. Lourdas, “Knx security overview,” Available at KNXSecurityoverview Accessed : 2019/11/12.

[203] ——, “Knx data secure,” Available at https://support.knx.org/hc/en-us/articles/360012689639-KNX-Data-Secure Accessed : 2019/11/12.