NTNU – Trondheim
Norwegian University of
Science and Technology

Authentication and Encryption in Janus-Based Wireless Underwater Communications

Branislav Petrović

Submission date:	June 2022
Supervisor:	Colin Boyd, NTNU IIK
Co-supervisor:	Bálint Zoltán Téglásy, NTNU ITK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	Authentication and Encryption in Janus-Based Wireless Underwater Communications
Student:	Branislav Petrović

Problem description:

Wireless underwater communications most commonly utilize acoustic waves as a signal carrier due to their superior robustness and range compared to radio waves or other transmission media in this environment. But, usage of acoustic waves introduces constraints as well, such as a low data rate and high packet loss. In addition, most information sent using acoustic waves under water today is unencrypted and unauthenticated, and it is assumed that all nodes in underwater communication networks are friendly. With the increasing exploration and development of underwater environments for economic benefit, environment preservation and research, there is an increased need for data protection among marine operators in this field, since the underwater threat landscape is rapidly broadening with new kinds of attacks, such as eavesdropping, routing attacks, and data tampering.

To overcome these problems, in this thesis, we shall propose security schemes integrated with the first standard for wireless acoustic underwater communication, Janus. The aim is to counteract the threats, bearing in mind the limitations of the acoustic communication channels. Novel schemes based on symmetric cryptography will be investigated, ultimately aiming to provide a way for underwater nodes to exchange authenticated and encrypted information. The outcome of this research will be a contribution to increasing the security of the emerging Internet of Underwater Things and its interoperability with the communication environment above the sea surface.

Date approved:	18.02.2022
Supervisor:	Colin Boyd, NTNU IIK
Cosupervisor:	Bálint Zoltán Téglásy, NTNU ITK

Abstract

Wireless underwater communications commonly utilize acoustic waves due to their superior robustness and range compared to radio waves or other transmission media. But usage of acoustic waves still suffers from high packet loss and it provides a low data rate. In addition, most information sent using acoustic waves under water today is unencrypted and not authenticated. With the development of underwater environments, there is an increased need for data protection among marine operators, since the underwater threat landscape is rapidly broadening with new kinds of attacks, such as eavesdropping, routing attacks, and data tampering.

To overcome these problems, in this thesis, we propose two security schemes integrated with the first standard for acoustic underwater communication, Janus. The aim is to counteract the threats, bearing in mind the limitations of the acoustic communication channels. The proposed schemes are based on symmetric cryptography. We base the work in this thesis on the first solution for mutual cryptographic authentication among underwater devices, proposed by Téglásy et al. The ultimate goal is to provide a way for underwater nodes to exchange authenticated and encrypted information.

We include a security analysis of the proposed security schemes based on a review of the general underwater threat landscape and general security requirements of underwater acoustic communication. The analysis shows that the schemes provide a high level of security against attacks on the proposed protocols and against tampering with the communication channel.

We simulate the proposed schemes by incorporating them in the existing implementation of Janus and the Underwater Acoustic Network (UAN) library of the Network Simulator 3 (NS3) network simulator. From the simulations, we show that the authentication protocols based on the proposed schemes complete within an acceptable time frame relative to the general communication requirements.

Sammendrag

Trådløs undervannskommunikasjon finner vanligvis sted ved hjelp av akustiske bølger på grunn av deres overlegne robusthet og rekkevidde sammenlignet med radiobølger eller andre kommunikasjonsmedier. Men akustiske bølger lider fortsatt av høyt pakketap og de gir en lav datahastighet. I tillegg er det meste av informasjon sendt ved hjelp av akustiske bølger under vann i dag ukryptert og ikke autentisert. Med den stadige utviklingen av undervannsmiljøer i dag, er det et økt behov for databeskyttelse blant marine operatører, siden trussellandskapet under vann utvides raskt med nye typer angrep, som avlytting, rutingangrep og tukling av data.

For å overvinne disse problemene foreslår vi i denne oppgaven to sikkerhetsløsninger integrert med den første standarden for akustisk undervannskommunikasjon, Janus. Målet er å motvirke truslene, med tanke på begrensningene til de akustiske kommunikasjonskanalene. De foreslåtte løsningene er basert på symmetrisk kryptografi. Vi baserer arbeidet i denne oppgaven på den første løsningen for gjensidig kryptografisk autentisering blant undervannsutstyr, foreslått av Téglásy m. fl. Det endelige målet er å muliggjøre det for undervannsnoder å utveksle autentisert og kryptert informasjon.

Vi inkluderer en sikkerhetsanalyse av de foreslåtte sikkerhetsløsningene basert på en gjennomgang av det generelle trussellandskapet under vann, samt generelle sikkerhetskrav til akustisk undervannskommunikasjon. Analysen viser at løsningene gir høy sikkerhet mot angrep på de foreslåtte protokollene og mot tukling med kommunikasjonskanalen.

Vi simulerer de foreslåtte løsningene ved å inkorporere dem i den eksisterende implementeringen av Janus og biblioteket for akustiske undervannsnettverk til nettverkssimulatoren NS3. Simuleringene viser at autentiseringsprotokollene basert på de foreslåtte ordningene fullføres innen en akseptabel tidsramme i forhold til de generelle kommunikasjonskravene.

Preface

This thesis serves as the final work in my 2-year Master of Science program in Communication Technology (MSTCNNS) at the Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisor, Colin Boyd, and my co-supervisor, Bálint Zoltán Téglásy for many valuable discussions and feedback, both during and outside scheduled meetings, as well as much relevant and useful material that proved valuable in the execution of this project.

Thanks also to professors Sokratis Katsikas and John Potter for additional input and feedback, as well as to Emil Wengle, for kindly lending real acoustic modems for investigating how the proposed security schemes would eventually be implemented in the future work.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

$\oplus$	Bitwise XOR.
&	Bitwise AND.
$\|\|$	String concatenation.
$\|X\|$	The bit length of an element X.
$\lceil X \rceil$	The ceiling function applied to the number X. Results in the next integer greater than X.

List of Acronyms

ACK Acknowledgement.

ADB Application Data Block.

AE Authenticated Encryption.

AEAD Authenticated Encryption with Associated Data.

AES Advanced Encryption Standard.

AI Artificial Intelligence.

AIS Automatic Identification System.

AUV Autonomous Underwater Vehicle.

BGP Border Gateway Protocol.

C2 Command and Control.

CAESAR Competition for Authenticated Encryption: Security, Applicability, and Robustness.

CBC Cipher Block Chaining.

CBC-MAC CBC-Message Authentication Code.

CCM Counter Mode with CBC-MAC.

CCMP CCM Protocol.

CRC Cyclic Redundancy Check.

CTR Counter.

DoS Denial of Service.

ECB Electronic Code Book.

ECDH Elliptic Curve Diffie-Hellman.

GCM Galois/Counter Mode.

GF Galois Field.

IoT Internet of Things.

IoUT Internet of Underwater Things.

IV Initialization Vector.

KDF Key Derivation Function.

LSB Least Significant Bit.

MAC Message Authentication Code.

MitM Man in the Middle.

MMPE Monterrey Miami Parabolic Equation.

MMSI Maritime Mobile Service Identity.

MSB Most Significant Bit.

NIST National Institute of Standards and Technology.

NS3 Network Simulator 3.

NTNU Norwegian University of Science and Technology.

OCB Offset Code Book Mode.

OSPF Open Shortest Path First.

PKI Public Key Infrastructure.

PRF Pseudo-Random Function.

PRP Pseudo-Random Permutation.

RBG Random Bit Generator.

ROS Robot Operating System.

RSN Robust Security Network.

RTT Round-Trip Time.

SDM Software Defined Modem.

SHA Secure Hash Algorithm.

SPN Substitution-Permutation Network.

SYN Synchronization.

TCP Transmission Control Protocol.

TKIP Temporal Key Integrity Protocol.

TUBcipher Tiny Underwater Block Cipher.

UAN Underwater Acoustic Network.

USBL Ultra Short Baseline.

UWCN Underwater Communication Network.

UWSN Underwater Wireless Sensor Network.

WEP Wired Equivalent Privacy.

WiFi Wireless Fidelity.

WLAN Wireless Local Area Network.

WPA Wireless Fidelity (WiFi) Protected Access.

WUCaN Wireless Underwater Communication and Networking.

XOR Exclusive OR.

Chapter 1 Introduction

Wireless underwater communication networks are rapidly increasing in quantity and there exist several types of underwater networks that serve different purposes. For example, Underwater Wireless Sensor Networks (UWSNs) consist of resource-constrained devices that are used for monitoring of underwater environments and infrastructure, and mostly send small amounts of data to a few destinations. Another kind of network consists of more advanced devices, such as Autonomous Underwater Vehicles (AUVs), that are not stationary and engage in more complex missions, such as reconnaissance for an organization with a stationary base. Communication and interoperability among the different network types may also be needed in certain operations. However, development of underwater communication standards is at an early stage. Currently, few standardized solutions for underwater communication exist and not all are available for public use. The currently most established digital underwater communication standard is Janus [PAG+14], a physical-layer, acoustic standard that is robust and makes interoperability among maritime operators possible. However, Janus by itself provides no security mechanism, neither in the form of encryption or authentication. On the other hand, the threat landscape related to this kind of communication has developed substantially in the recent years[1].

In this introductory chapter, the motivation for the work in this project is explained and the main challenges are presented. Then, the research questions and objectives are listed and the contributions of the thesis are explained. Finally, the outline of the whole thesis is given.

1.1 Motivation

The proliferation of underwater communication means is encouraging devices to transmit steadily increasing amounts of data for different purposes and for different operators. Typical examples of data transferred in Underwater Communication

[1]This threat landscape is presented thoroughly in Section 2.4

Networks (UWCNs) are meteorological and geophysical readings, mission-critical Command and Control (C2) and signal messages, as well as information about classified military operations. Some of this data is regarded sensitive, as attackers with an interest to disrupt operators' underwater missions can obtain information that should be kept secret, or perform other manipulations. With the increased amount of such sensitive data exchanged under water, strong security mechanisms that protect the confidentiality and integrity of the data are a natural need. As the Janus standard already serves as a basis for wireless underwater communication, in this thesis, we focus on adding security mechanisms to it, thereby contributing to the establishment of a standard by which marine operators can share the same waters and cooperate. Which security mechanisms should be added and how they should function depends on what relationships the communicating parties have to each other, what data they need to exchange, and the capabilities of the adversaries that can disrupt the communication. Authentication of messages and entities is an important requirement, in addition to encryption, since not only confidentiality, but also integrity of data is to be ensured. If authentication and encryption are implemented separately, they may be too slow for providing security in underwater communication. An alternative approach is to use so-called authenticated encryption, which is a more efficient mechanism that provides both integrity and confidentiality. Therefore, in this thesis, we focus primarily on this approach to providing both integrity and confidentiality of data exchanged in Janus-based communication.

1.2 Main Challenges

The main constraints for providing security features under water are the following:

- **Low data rate:** A much lower data rate can be achieved with acoustic underwater communication than with radio waves above the sea surface. This causes significant delays in data transmission, which increases the time spent by security protocols.
- **High packet loss:** High packet loss in comparison to air interfaces is caused by attenuation and fading when transmitting through water, as well as by Denial of Service (DoS), as a consequence of, for example, jamming. This can prevent security protocols from completing successfully, resulting in reduced availability.

The above constraints make the use of many standardized encryption and authentication schemes infeasible due to their complexity and scale. Even solutions from other resource-constrained areas, such as the Internet of Things (IoT) above water would add a significant overhead to the communication. Therefore, authenticated encryption mechanisms used under water must be computationally inexpensive and reduce the communication overhead to a minimum. Thus, designing such mechanisms is an extremely challenging task, which requires both technical compromises and a thorough theoretical security analysis of the proposed solutions.

1.3 Research Questions and Objectives

Bearing in mind the motivation and main challenges exposed above, we define the following research questions for this thesis.

RQ1: How can authenticated encryption mechanisms be applied in Janus-based underwater communication to provide integrity and confidentiality of data?

RQ2: How can the underwater authenticated encryption schemes (ref. RQ1) be made to complete reliably within a required time?

RQ3: What level of security do the proposed schemes provide in relation to the underwater threat landscape?

To answer the research questions, the following objectives are pursued:

OBJ1: Identify and apply suitable Authenticated Encryption with Associated Data (AEAD) schemes for use with Janus.

OBJ2: Implement the chosen schemes into the Janus simulation and simulate them together with Janus.

OBJ3: Measure the time needed for the chosen schemes to complete in the simulated environment, with and without noise.

OBJ4: Make an overview of the underwater threat landscape and identify vulnerabilities of the chosen schemes in relation to the threat landscape.

1.4 Contributions of the Thesis

The main contribution of this thesis is the incorporation of two authenticated encryption schemes, CCM [WHF02] and AEGIS [WP13], in Janus. This involved numerous technical interventions in the original specifications of CCM and AEGIS. However, the core functionality of these algorithms has been kept, such that they provide their original levels of security. This is the first time that such AEAD schemes have been proposed for use with this standard. We have defined authentication protocols utilizing these schemes in Janus, both with and without so-called cargo packets. In addition, we have performed comparative analysis of security, performance, and scalability of the new proposed solutions and the original solution proposed by Téglásy et al. [TWPK]. Ultimately, this thesis lays the ground for future investigation of the suitability of authenticated encryption and other security mechanisms in Janus-based underwater communication.

1.5 Outline of the Thesis

The thesis is structured as follows: Chapter 2 gives an overview of both general and Janus-based underwater communication, as well as the threats posed to such communication. It also discusses what security properties are needed with respect to the threat landscape and presents related work that has been done in this area. Chapter 3 presents CCM and AEGIS in detail, gives arguments in favor of their use for our purpose and explains how they can be incorporated into Janus. Chapter 4 analyzes the security of the schemes proposed in Chapter 3 with respect to the reviewed underwater threat landscape. Chapter 5 gives an overview of the methodology of the work conducted in the thesis with respect to design science and traditional science. Chapter 6 describes how the experimental work in the thesis was conducted and it presents and discusses the results obtained through this work. Chapter 7 concludes the thesis and discusses how the work can be continued and further improved.

Chapter 2
Background

In this chapter, background material needed to understand the authentication protocols presented in subsequent chapters, is presented. First, a description of underwater communication is provided, which discusses possible transmission media, general operation of UANs, as well as the Janus standard. Then, general threats to underwater communication and the corresponding security requirements are discussed. Afterwards, an introduction to cryptographic concepts, such as Authenticated Encryption (AE), Substitution-Permutation Networks (SPNs), and block cipher modes of operation, is given. Finally, the state of the art of the work on underwater security, as well as other related work is presented.

2.1 Transmission Media

Even though sound is the most widespread transmission medium for underwater communication, it is not the only available option in this environment. The fact that acoustic waves are the least susceptible to attenuation by water particles makes them the most widespread option with the greatest potential for application in standards. The biggest drawback of this medium is the extremely low data rate that it provides compared to other media.

Electromagnetic radio waves that are used in the Internet above the sea surface can also be used to transmit data underwater, as shown by Lloret et al. [LSAR12]. In this work, the 2.4 GHz band from the IEEE 802.11b/g standards is submerged in freshwater to transmit data at 11 Mbps. Radio waves would be the preferred option for the development of a bridge interface between the Internet and underwater networks. However, as shown by Lloret et al., even though much higher data rates can be achieved with this medium compared to acoustics, only communication at extremely close range can be achieved without severe attenuation in water, with packet loss increasing drastically above a distance of only 15 cm.

Another possible medium are free-space optical waves, that are normally used in

Table 2.1: Physical transmission media for underwater communication. Adapted from [TWPK].

Medium	Data rate	Maximum range without data loss
Electromagnetic	11 Mbps	15 cm
Free-space optical	500 Mbps	150 m
Acoustic	80 bps	10 km

optical fibres, but propagating directly through water instead of a cable. Wang et al. [WLLX19] have achieved a data rate of 500 Mbps at a distance of 146 m through clean freshwater under laboratory conditions using laser diodes with a wavelength of 520 nm. While this medium provides a very high data rate, the optical waves are still highly attenuated by water, and fading occurs after a relatively short distance, causing data loss and a decreased data rate. This can be seen in the same work by Wang et al., where an increase of the distance by approximately 30 m led to a drop in the data rate to 100 Mbps.

A summary of the properties of the different transmission media is shown in Table 2.1 (Téglásy et al. [TWPK]). Ultimately, acoustic communication stands out as the medium that provides the only acceptable range and reliability for practical use outside laboratory conditions. For this reason, there is much more research conducted on the propagation of sound through water, sound modulation, etc., than electromagnetic or optical waves, and the use of acoustics is much more widespread in current UWCNs. This has an effect on the overall nature of underwater communications. The low data rate of acoustic waves makes it infeasible to exchange large amounts of data under water, meaning that devices are limited to communicating specific kinds of messages that are mostly used for mission-critical signalling. This provokes the need to carefully specify the purpose and meaning of all data transferred under water, and to thoughtfully choose the algorithms for networking and cryptography that are to be deployed.

2.2 General Operation of Underwater Acoustic Networks

A unique characteristic of UANs is the long propagation time of acoustic signals among devices, caused by the speed of sound through water. Because of this, a major feature of such networks is the possibility to determine the distance among devices and their positions relative to each other, based on the propagation time of the exchanged acoustic signals. Because of the lack of underwater infrastructure, such as large-scale servers with many clients found in the Internet, this is the only way of obtaining an overview of the status and positions of vessels in underwater networks. Typically, a transceiver mounted on a stationary base, such as an oil valve

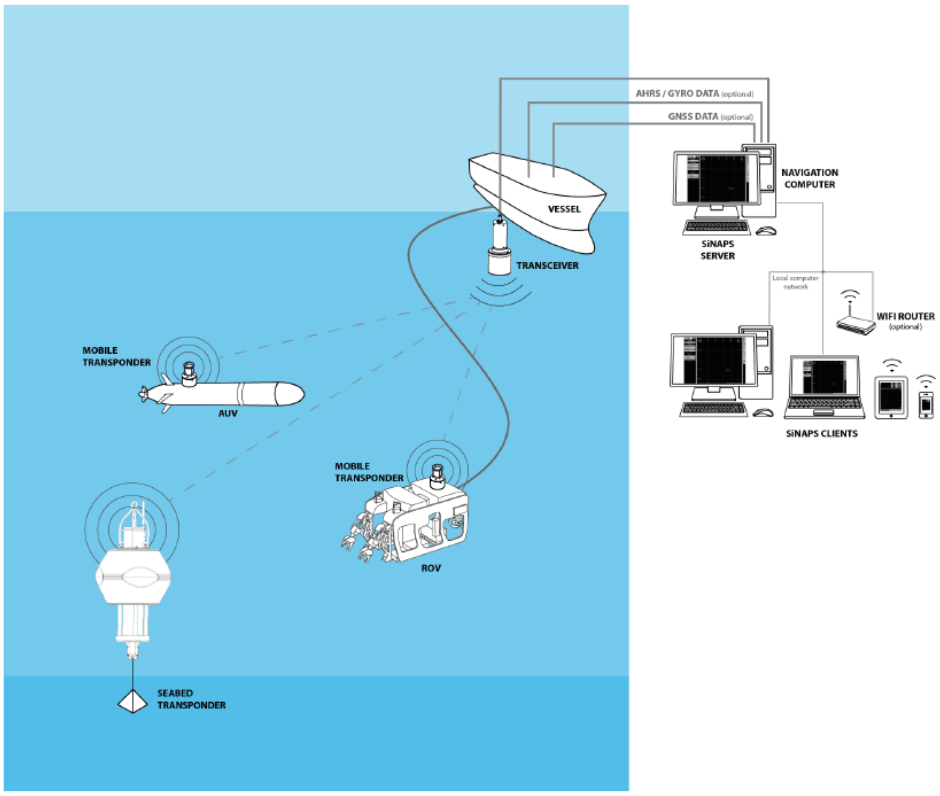


Figure 2.1: An example of a UAN with a stationary base and mobile devices. Source: [Evo18].

or a large ship, sends interrogation signals to underwater devices in the area. The underwater devices are equipped with transponders that detect the interrogations and respond with their own acoustic signals. Based on the time of propagation of these signals, the operation center on the base can calculate the positions of the underwater devices and process any information they send back. An example of a UAN in the form of an Ultra Short Baseline (USBL) positioning system (EvoLogics [Evo18]) is shown in Figure 2.1.

2.3 Janus

As the first digital underwater communication standard, the aim of Janus is to provide the means for different maritime operators to communicate and cooperate, regardless of the developer of their digital coding technologies. Before the publication of Janus in 2014, underwater communication capabilities were manufacturer-specific, with no universal standard by which operators supplied by different manufacturers could communicate. This would be an ever-increasing problem since the range of devices employed in underwater assets is broadening, creating a more heterogeneous communication environment. With the development of battery technology, processing and memory capabilities, as well as control theory and Artificial Intelligence (AI), there are many kinds of devices that can be employed for many different kinds of underwater operations. Janus provides the means to perform Wireless Underwater Communication and Networking (WUCaN) among devices with different purposes and by different manufacturers, to allow for a wider range and scale of operations that can be performed on underwater missions.

The Janus standard allows for a bandwidth of 80 bps and a range of 10 km (see Table 2.1). The baseline packet is 64 bits long. It is shown in Table 2.2. 34 of these bits (bits 23 to 56) are reserved for user data, and are called the Application Data Block (ADB). The rest of the packet consists of communication overhead that specifies different communication properties. The very limited amount of user data that can be transmitted in a packet makes Janus best suited towards communication with small data exchanges, such as C2 and status messages. If larger amounts of data are to be sent, it must be either distributed into several packets or be sent as a cargo immediately following a packet. If several packets are used, the additional overhead must be encoded and modulated for each packet, resulting in additional use of computing resources and increased latency. If a cargo is specified, the channel is reserved during the transmission of the cargo and no other communication is to happen on the network while it is being transmitted.

A cargo packet can be specified by setting the Schedule flag (bit 6) to 1 and the first bit of the ADB (bit 23) to 0 in the baseline packet. This way, any kind of user data can be sent in addition to the data in the ADB. The time to be reserved for the cargo is indicated by the subsequent 7 bits in the ADB (bits 24 to 30). Thus, the maximum amount of time that the channel can be reserved is obtained by setting these bits to a value of 127 (1111111_2). This will reserve the channel for 10 minutes, allowing to send a total cargo of 48000 bits. The structure of a baseline packet with optional cargo, with an indication of how many bits each field takes, is shown in Table 2.3.

2.4 Threat Landscape

The threats to underwater acoustic communication are similar to those faced by radio communication above the sea surface. In both cases, wireless transmission with physical waves is used, making it possible to use similar techniques to disrupt the propagation of the waves and, thereby, also the communication. In addition, routing protocols that forward packets based on channel quality and the possibility to reach adjacent nodes, are also in use in underwater networks. These protocols

Table 2.2: The bit allocation table of a Janus baseline packet. Adapted from [PAG+14].

Bits	**Descriptor**	**Values**	**Comments**
1 - 4	Version	0011	Unsigned 4 bit integer. Current version is 3.
5	Mobility flag	0=static, 1=mobile	Indicates nature of the transmitting platform.
6	Schedule flag	0=off, 1=on	If On (1), the first bit in the ADB indicates whether time reservation or a repeat interval is used.
7	Tx/Rx Flag	0=Tx-only, 1=Tx/Rx	Transmit/Receive capability.
8	Forward capability	0=no, 1=yes	Used for routing and Delay Tolerant Networking.
9-16	Class user i.d.	[00000000 : 11111111]	Allows 256 classes of users, mostly individual nations. Class 17 indicates use of Venilia.
17-22	Application Type	[000000 : 111111]	Allows 64 different types of message per user i.d. class.
23-56	Application Data Block	Determined by user	34-bit payload.
57-64	8-bit Checksum		8-bit CRC run on the previous 56 bits with polynomial $p(x) = x^8 + x^2 + x + 1; init = 0$.

Table 2.3: Structure of a Janus packet.

Version number	Mobility flag	Schedule flag	TX/Rx	Forward capability	Class user i.d.
4	1	1	1	1	8

Application type	Repeat	Reservation time	Application data block	CRC	Optional cargo
6	1	7	26	8	$n \leq 48000$

are equivalent to, for instance, Open Shortest Path First (OSPF) and the Border Gateway Protocol (BGP), used in the Internet today. Protocols of this kind can be manipulated by malicious users to cause disruptions in networks.

Ghannadrezaii and Bousquet [GB18] classify threats to underwater communication into three main categories:

- **Eavesdropping:** Eavesdropping is generally a passive attack in which an adversary intercepts data over a channel and tries to recover the information from it. In wireless radio communication above the sea surface, the interception is done by tuning a receiver to frequencies used for communication. This kind of interception is also fully possible under water, in which case the adversary would overhear the acoustic spectrum instead of the electromagnetic one.
- **Routing attacks:** Routing attacks are attacks against routing protocols, the kind of which is mentioned above. In these attacks, a malicious user selectively

directs packets via alternative routes, thereby disrupting network traffic. This can lead to the creation of routing loops or wormholes. An example of such an attack is described in Section 2.4.2.

– **Data tampering:** Data tampering involves attacks that consist of changing the contents of a packet as it is forwarded over the network. This can be done, for example, by a malicious node in the network that has intercepted the packet by eavesdropping. The node can then flip arbitrary bits in the packet and change the packet's information. This way, an attack that involves data injection, such as a replay attack with nonce reuse, can be initiated.

Domingo [Dom11] gives an overview of threats to WUCaN, possible counteractions of them, and open research areas related to security in this environment. Three major attacks that fall under the categories above are the jamming, wormhole, and Sybil attacks.

2.4.1 Jamming

A common tampering attack in UANs is jamming. Generally, UANs are more vulnerable to jamming and interference than their radio-based counterparts above the sea surface. This is due to the narrow acoustic frequency bands, which range from a few to a few hundreds of kHz. When jamming an underwater signal, an attacker would install carriers on the signal path between legitimate network nodes and transmit with the same frequency as the nodes (i.e., make noise). This causes interference with the legitimate acoustic signal, which leads to the disruption of communication due to packet loss. If the attacker in addition intercepts the messages that are dropped, the messages can later be replayed to the receiving node with incorrect timestamps or localization information, thereby causing a loss of synchronization of these values among the network nodes. The jamming attack is shown in Figure 2.2.

2.4.2 Wormhole Attack

Another serious threat to underwater communication is the Wormhole attack. It is performed by establishing an out-of-band connection between two legitimate nodes that are many hops away from each other in a network. The added connection may be created above the sea surface or through a wired link, such that the bandwidth is much higher and the propagation delay is much lower than in ordinary acoustic channels. This will cause routing protocols operating in a UAN to prefer this link due to its higher quality. The consequence is that a false neighbour relationship will be created between the legitimate nodes connected with the wormhole link deployed by the attacker. Thus, the legitimate nodes may exchange critical data across the wormhole link, giving the attacker the possibility to monitor, modify, or drop all packets. The attack is shown in Figure 2.3.

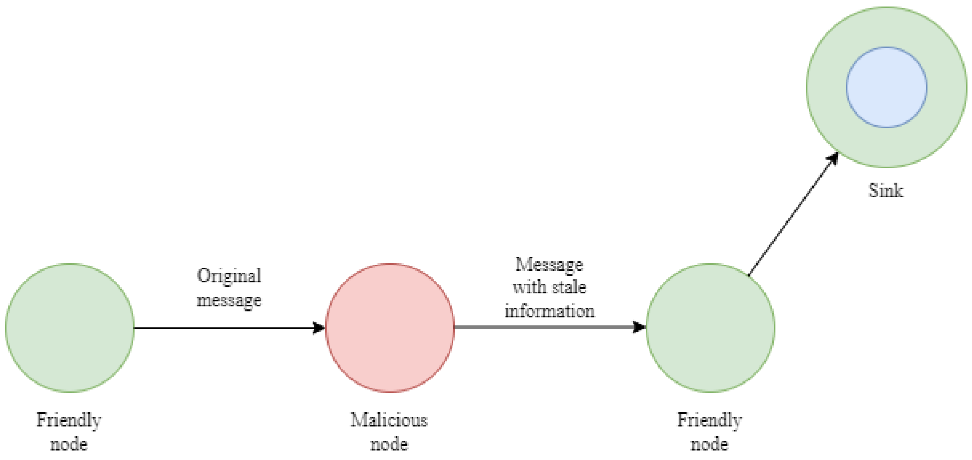


Figure 2.2: The Replay Attack set up by Jamming in UANs. Adapted from [Dom11].

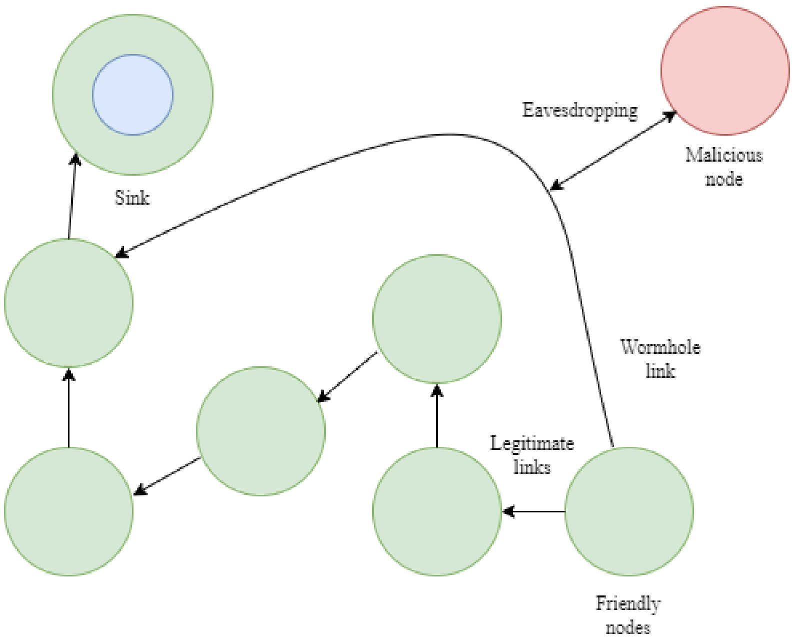


Figure 2.3: The Wormhole Attack in UANs. Adapted from [Dom11].

2.4.3 Sybil Attack

One possible attack in UWSNs, where data is forwarded with several hops to a certain destination, is the Sybil attack. Here, a malicious network node that has acquired multiple identities advertises the identities as valid, but nonexistent nodes. Then, when a legitimate node tries to forward or send data across the network, for example to a sink node, it may choose to send the data to the spoofed identities of the malicious node. The malicious node then intercepts the data by overhearing it. The attack is shown in Figure 2.4.

Many other kinds of attacks are possible, such as sinkhole attacks, acknowledgement spoofing, hello flood attacks, and selective forwarding. Hence, if the communication is not protected with appropriate security measures, it is extremely vulnerable.

2.5 Security Requirements in Underwater Communication

With respect to the threat landscape presented in Section 2.4, the following basic security properties are needed in UANs in general. The identification and definition of the security requirements was done in the preparation project preceding this thesis [PTB21]. The requirements have remained unchanged, and their presentation from the project report is included below. The properties themselves are described by Ryan et al. [RSG+00, p. 6-14] and in Recommendation X.800 by the International Telecommunication Union (ITU) [TC91]. In addition to explaining the properties themselves, we explain how they, if satisfied, provide protection against the specific attacks described in Section 2.4.

2.5.1 Authentication of origin and entity authentication

Authentication is needed as proof that the data was sent by a legitimate sender. It is essential in military and safety-critical applications of UWCNs. Two types of authentication are authentication of origin and entity authentication. These types are strongly related. Authentication of origin means that one should be certain that a message that is claimed to be sent from a certain party is indeed sent from that party. Entity authentication is the process of providing assurance about the identity of a party interacting with a system (e.g., to access a resource). In other words, authentication of origin is authentication of messages, while entity authentication is the authentication of communicating parties while interacting with a system or with each other. Both forms can be needed in UWCNs, depending on the application. With respect to the threat landscape from Section 2.4, authentication of origin and entities would protect against the Wormhole and Sybil attacks.

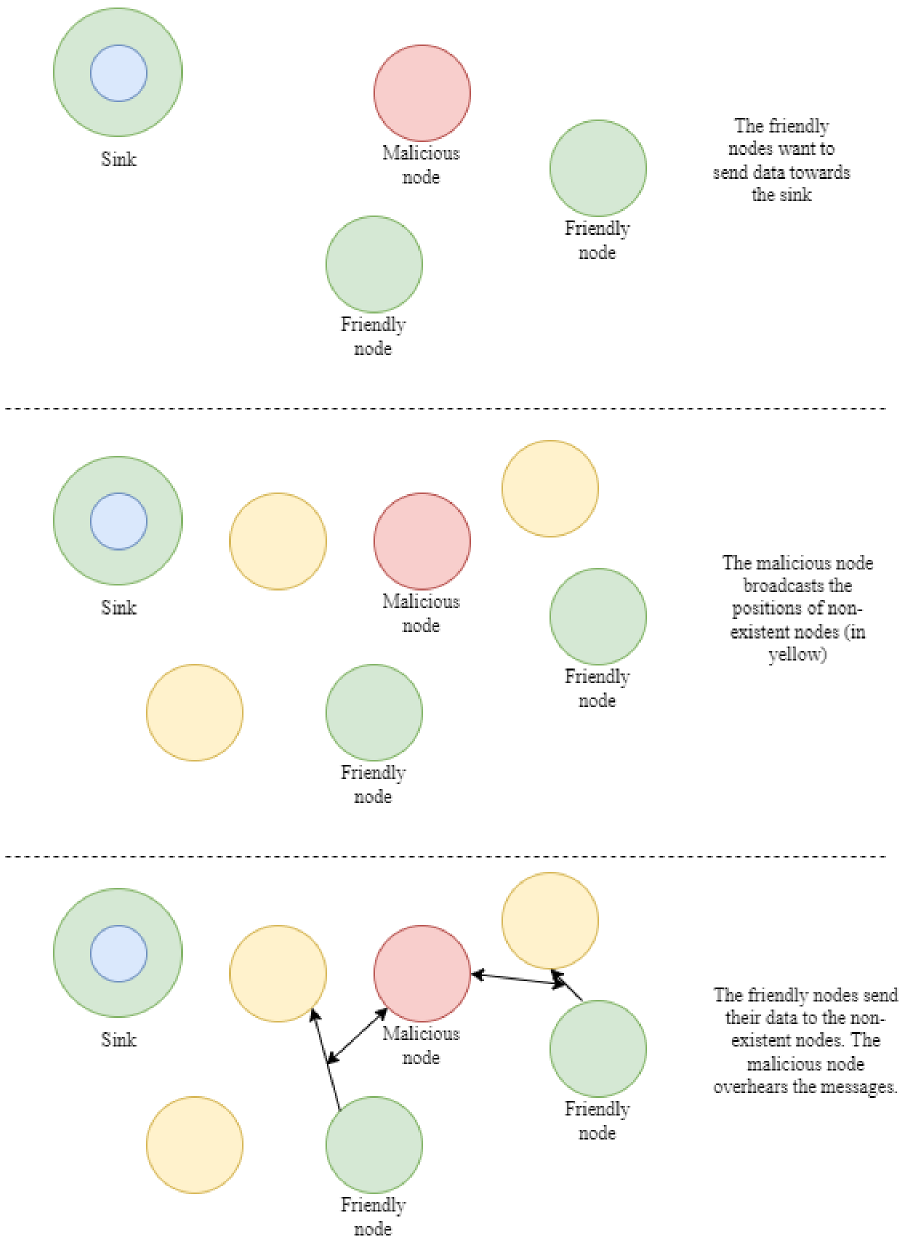

Sink
Malicious node
Friendly node
Friendly node
The friendly nodes want to send data towards the sink
Sink
Malicious node
Friendly node
Friendly node
The malicious node broadcasts the positions of non-existent nodes (in yellow)
Sink
Malicious node
Friendly node
Friendly node
The friendly nodes send their data to the non-existent nodes. The malicious node overhears the messages.

Figure 2.4: The Sybil Attack in UANs. Adapted from [Dom11].

2.5.2 Confidentiality

Confidentiality or secrecy means that information should not be available to unauthorized third parties. Typically, this is achieved with encryption, in which case it is expected that an adversary will intercept and obtain messages between legitimate parties and read meta-information, but it will not be possible to decrypt the message. If there is a set of sensitive pieces of information to protect, then the confidentiality property is not satisfied if an adversary discloses any of the pieces. This in general means that the adversary should not obtain the decryption keys. C2 and surveillance messages in mission-critical operations in UWCNs should be confidential and therefore encrypted. Adversaries that intercept packets, for instance in Wormhole attacks, may disclose confidential information if it is not encrypted.

2.5.3 Integrity

Integrity is strongly tied with authentication of origin in the sense that it is required that the contents of the output message match that of the input message. It generally means that data cannot be altered on the communication path, or that any alteration will be noticed and corrected. Integrity is a property that directly addresses data tampering attacks described in Section 2.4. As an example, there is a need for integrity of information in underwater sensor applications for environmental preservation, such as water quality monitoring.

2.5.4 Availability

Related to a protocol, availability means that it should be able to be relied upon to achieve a certain goal. For example, in a key exchange protocol, one should be certain that the key exchange will happen, and that messages will not be erased in transmission by adversaries. Availability also means that data and services should be available when needed, and be resistant against DoS attacks. Lack of availability of this kind in UWCNs would especially affect time-critical aquatic exploration applications such as prediction of sub-sea earthquakes.

2.6 Fundamentals of Symmetric Ciphers

In this section, we briefly explain block ciphers and their basic modes of operation, as well as stream ciphers, since these structures constitute basic elements of the mechanisms that provide confidentiality and integrity of data. In addition, SPNs are explained as the basic round structures for modern block ciphers, such as the Advanced Encryption Standard (AES).

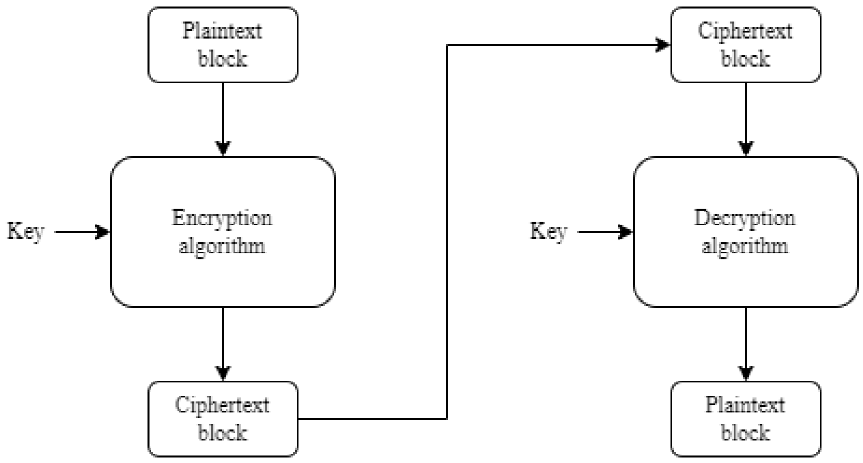


Figure 2.5: General overview of a block cipher. Adapted from [Sta17].

2.6.1 Block Ciphers

A block cipher is a cipher that treats a block of plaintext (i.e., a set of plaintext symbols of a fixed size) as a whole and produces a ciphertext of equal length (see, for example, [Sta17]). Typically, the size of a block is between 64 and 256 bits. Each block is encrypted with the same key. The key is used both for encryption and decryption, and both parties that exchange confidential data share the same key. The basic concept of a block cipher is shown in Figure 2.5.

2.6.2 Stream Ciphers

In a stream cipher, the plaintext is encrypted one symbol at a time (typically, a bit or a byte, etc.). For this, a keystream of the required length is used. As with block ciphers, the keystream must be shared by all parties willing to exchange confidential data, or they must be able to generate the same keystream given equal initialization values, such as a generating key. The plaintext bits are XORed with the keystream bits to produce the ciphertext bits, and the ciphertext bits are XORed with the same keystream bits at the receiving side to re-obtain the plaintext bits. It is important that the keystream has good randomness properties, such that it is infeasible to predict future portions of the keystream based on previous ones. A general stream cipher structure is shown in Figure 2.6.

2.6.3 Modes of Operation for Block Ciphers

Block ciphers can be specified to function in certain modes of operation, also called confidentiality modes. Common confidentiality modes are specified in Special Publication 800-38A of the National Institute of Standards and Technology (NIST) [Dwo01].

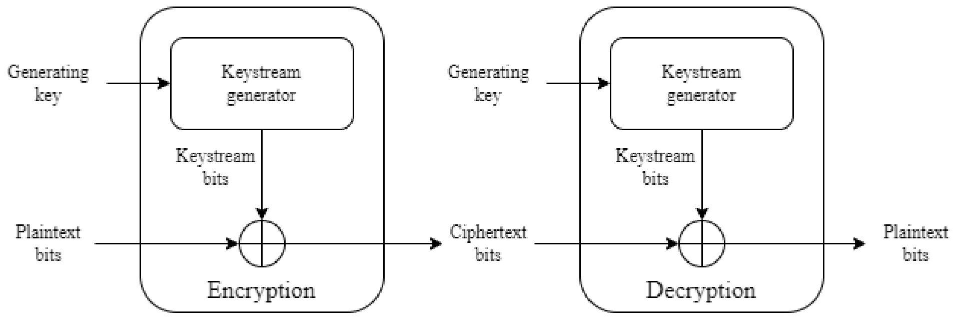


Figure 2.6: General overview of a stream cipher. Adapted from [Sta17].

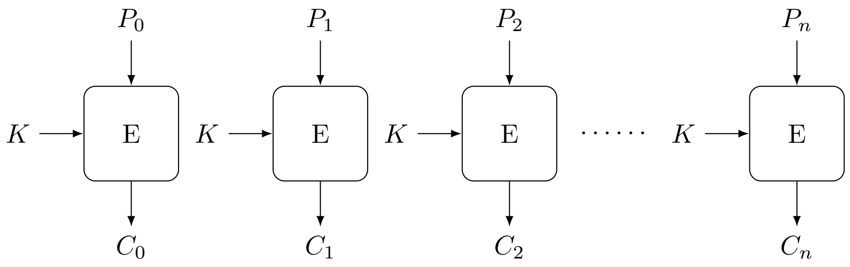


Figure 2.7: Overview of ECB mode encryption. Adapted from [Jea16].

The relevant modes for this thesis are the Electronic Code Book (ECB), Cipher Block Chaining (CBC), and Counter (CTR) modes. They are described below.

ECB Mode

ECB is the basic mode for a block cipher. For a given key, a fixed ciphertext block is assigned to each plaintext block, and all executions of the block cipher are independent of each other. In other words, each ciphertext block is described as $C_i = E_K(P_i)$ for $1 \leq i \leq n$, where n is the number of blocks and E is the block cipher in the forward direction. Similarly, the decryption process that reproduces the plaintext blocks is described as $P_i = E_K^{-1}(C_i)$, where E^{-1} is the inverse block cipher. A general overview of encryption in ECB mode is given in Figure 2.7.

CBC Mode

CBC mode involves combining plaintext blocks with previous ciphertext blocks. An Initialization Vector (IV) is combined with the first plaintext block. The IV must be unpredictable and each IV must only be used once with a given key. The plaintext blocks are combined with previous ciphertext blocks by applying the Exclusive OR (XOR) operation, before encrypting the resulting string with the block cipher in

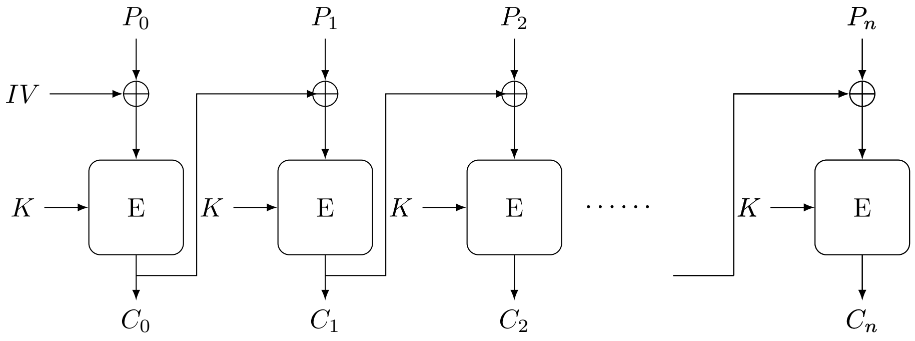


Figure 2.8: Overview of CBC mode encryption. Adapted from [Jea16].

the forward direction. The same key is used to encrypt all blocks. Formally, each ciphertext block can be represented as

$$C_i = E_K(P_i \oplus C_{i-1}), 1 \leq i \leq n, \tag{2.1}$$

where n is the number of blocks. Similarly, the ciphertext is decrypted by executing the block cipher in the inverse direction, starting from the first ciphertext block and XORing it with the decryption of the next ciphertext block. The same IV that was used in encryption is used to obtain the first plaintext block. Formally,

$$P_i = E_K^{-1}(C_i) \oplus C_{i-1}, 1 \leq i \leq n. \tag{2.2}$$

A general overview of encryption in CBC mode is given in Figure 2.8. In CBC mode, the block cipher remains a block cipher (it is not converted into a stream cipher). Therefore, if the last block to be encrypted is incomplete, it must be padded such that the entire block is occupied.

CTR Mode

A block cipher in CTR mode is a synchronous stream cipher, meaning that the keystream is generated independently of the plaintext. It is generated by encrypting successive values of a counter, concatenated with an initialization value, called a nonce, N. For each block, each keystream bit is then XORed with its corresponding plaintext bit, as shown in Figure 2.6. All counters must be unique in each sequence of blocks that is processed. The keystream for the block i is generated as follows:

$$O_i = E_K(N||i), 1 \leq i \leq n, \tag{2.3}$$

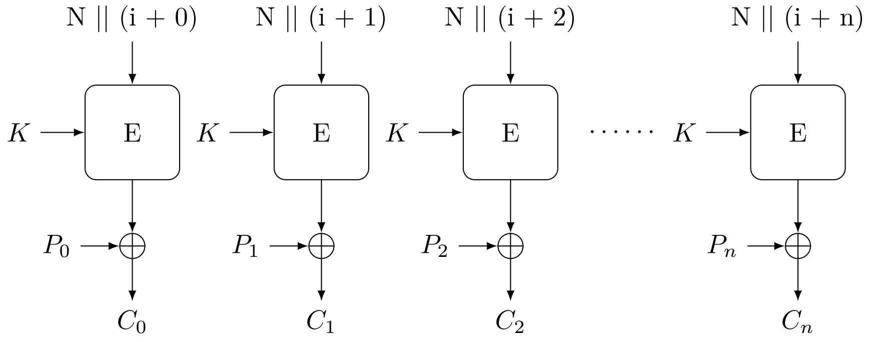


Figure 2.9: Overview of CTR mode encryption. Adapted from [Jea16].

where n is the number of blocks. The keystream is then used to produce each ciphertext block in this manner:

$$C_i = O_i \oplus P_i. \tag{2.4}$$

Similarly, decryption is done by generating the keystream blocks once more and XORing them with the ciphertext bits:

$$P_i = O_i \oplus C_i. \tag{2.5}$$

It is worth noting that a block cipher in CTR mode operates only in the forward direction for both encryption and decryption. A general overview of encryption in CTR mode is illustrated in Figure 2.9.

2.6.4 Substitution-Permutation Networks (SPNs)

SPNs are a type of iterated cipher, which again are a type of product cipher, which provide the basic properties of confusion and diffusion, proposed by Shannon [Sha49]. Confusion is achieved by substitution, which makes the relationship between the key and the ciphertext as complex (nonlinear) as possible. Diffusion is achieved by permutation, which dissipates the statistical properties of the plaintext across the ciphertext. In modern SPNs, these stages are performed multiple times, in rounds.

Generally, a plaintext block in an SPN is split into smaller blocks. If the original block length is n, then the block should be split into m smaller blocks of length l, such that $n = lm$. First, the round key corresponding to the current round is XORed with the entire original block. The substitution stage is then performed by substituting each smaller block of length l by another block of equal length. This is

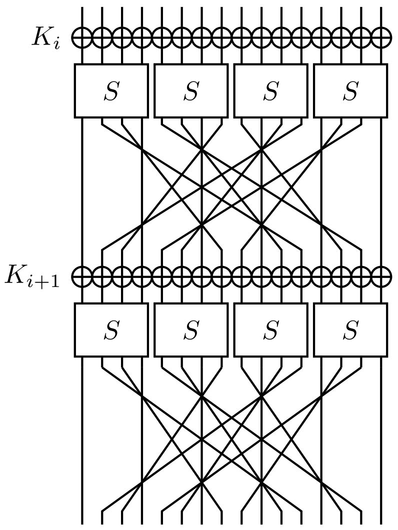


Figure 2.10: Two rounds of an SPN. First, the round key is applied, before the substitution S and the permutation. The use of a new round key indicates the start of a new round. Adapted from [Jea16].

usually done by looking up each block and its corresponding substitution in a table, normally called an S-box. The blocks are usually binary strings, so the substitution function is the mapping $\{0,1\}^l \to \{0,1\}^l$. The permutation stage then follows, which transposes all sub-blocks of the original block among themselves. It is thus the mapping $\{1,2,...,n\} \to \{1,2,...,n\}$. After this stage, the next round follows under the next round key. A general overview of an SPN is shown in Figure 2.10.

2.7 Authenticated Encryption

Authenticated encryption is a type of construction that combines separate algorithms that independently provide encryption and authentication into one. Algorithms that typically provide encryption are ciphers, which are generally distinguished as block ciphers and stream ciphers. A method of providing authentication is a Message Authentication Code (MAC), which ensures that a message originated from a legitimate party and has not been tampered with during transmission.

Wu and Preneel [WP13] present three ways of combining authentication and encryption algorithms. Firstly, a block cipher may be used in a special mode of operation and adapted to encrypt a message and produce a MAC tag in the same operation. The second presented method is to use a stream cipher, where the keystream is split into two parts: one part for encryption and one for authentication. Finally, a dedicated authenticated encryption algorithm may be utilized, which uses a message to update the state of a cipher (either a block cipher or a stream cipher). This approach is generally considered the most efficient, as it involves designing algorithms specifically for authenticated encryption and it does not require large adaptations of independent authentication and encryption algorithms. In this thesis, we focus on the first and the third option, namely a mode of operation for a block cipher (CCM) and a dedicated algorithm (AEGIS).

2.8 State of The Art of Underwater Security

In this section, we review the status of solutions that provide integrity and confidentiality in underwater communication. Most notably, we present the first ever authentication solution for this environment, proposed by Téglásy et al., on which we base our solutions. Also, we present an encryption scheme made specifically for Janus, Venilia.

2.8.1 Authentication of Underwater Assets

The first proposal for an authentication procedure based on Janus is given by Téglásy et al. [TWPK]. Here, two devices, initially unknown to each other, first identify each other as friend or foe by determining whether they possess the same pre-shared key. They achieve this by exchanging a timestamp TS, a clock accuracy descriptor CD, and two Synchronization (SYN) and Acknowledgement (ACK) flags in the ADB of the Janus packet. The timestamp and the clock accuracy descriptor are encrypted with the 32-bit block version of the RC5 cipher [Riv95] using a pre-shared long-term key K_n of at least 128 bits in length. The flags are not encrypted. The authentication is based on the validity of the timestamps exchanged by the devices and the assumption that the sending device would be unable to encrypt the message without K_n. Assuming that the devices' clocks were synchronized during the exchange of K_n at the start of the mission, a device checks if the timestamp it received is within the expected bounds compared to the mission duration. The expected bounds are adjusted according to possible deviations in clock synchronisation between the devices and the expected maximum distance that a message can travel. If the timestamp is valid, the receiver sends its own timestamp and clock accuracy descriptor back to the originating device, encrypted with the same key K_n. The timestamps also allow the performance of ranging i.e., determining the distance between the authenticating devices. This is done by comparing the timestamps that are exchanged and performing simple calculations based on the speed of sound. Thus, the general properties of a UAN, as described in Section 2.2, are achieved. The protocol for recognition of friendly devices is shown in Figure 2.11.

To protect the communication in the case of compromise of K_n, the exchanged values are later used to compute a session key K_{AB} with a custom Key Derivation

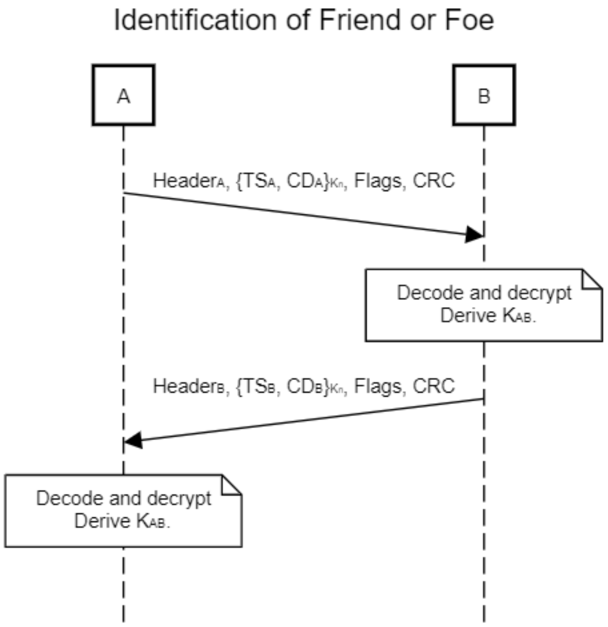

Identification of Friend or Foe
A
B
HeaderA, {TSA, CDA}Kn, Flags, CRC
Decode and decrypt
Derive KAB.
HeaderB, {TSB, CDB}Kn, Flags, CRC
Decode and decrypt
Derive KAB.

Figure 2.11: The protocol for identification of friend or foe with timestamps. Adapted from [TWPK].

Function (KDF). The KDF also utilizes RC5, albeit with a block size of 128 bits, to produce a 256-bit ciphertext. K_{AB} allows communication to remain secure if K_n is compromised after the establishment of K_{AB} since subsequent messages will be encrypted with K_{AB} instead of K_n. It is also infeasible for attackers to obtain previous session keys if they obtain K_n and they have not eavesdropped on previous runs of the protocol. Since K_{AB} is derived from a timestamp, a new timestamp sent by an attacker to a legitimate node as an authentication attempt will result in a different key due to the passing of time and, thus, a different timestamp.

As one device may derive session keys with many other devices, it is necessary to create a mapping of the derived session keys to their corresponding device identities. This is achieved with a shared lookup table at each device, containing all other devices' identities with their corresponding long-term keys. When a device receives a message that is encrypted with a certain long-term key, it will attempt to decrypt the message with the long-term keys in its lookup table, sequentially. If one of the keys yields a successful decryption, the key that yielded the decryption is used with the KDF to derive a session key, and the session key is stored along with the identity and the long-term key in the table. The identity has the form of a Maritime Mobile Service Identity (MMSI), which is a standard format in the Automatic Identification System (AIS), used for tracking ships above the sea surface. The length of the MMSI

Table 2.4: An example lookup table with MMSIs, long-term keys, and session keys on device A. Messages encrypted with K_{nD}, K_{nF}, and K_{nG} have been received. The network consists of 7 devices, including A.

MMSI$_B$	K_{nB}	
MMSI$_C$	K_{nC}	
MMSI$_D$	K_{nD}	K_{AD}
MMSI$_E$	K_{nE}	
MMSI$_F$	K_{nF}	K_{AF}
MMSI$_G$	K_{nG}	K_{AG}

is 30 bits. An example of a lookup table of this kind is shown in Table 2.4.

Limitations

Although this protocol provides authentication, encryption, and key establishment, it has certain limitations. They are outlined below. The protocols that are proposed in this thesis aim to address these limitations.

Timestamp forgery and replay Authentication of messages that relies solely on the validity of timestamps is vulnerable since timestamps can be forged by an attacker relatively simply. If attackers obtain the long-term key of one or more devices in the network and those long-term keys have not yet been used to derive corresponding session keys, the attackers can send their own TS and CD values to a legitimate device as an authentication attempt. TS and CD do not require any secret values to generate and, as such, they can be generated by any device with an onboard clock. As long as the attackers used a valid K_n to encrypt their message, they will be able to successfully establish a session key with a valid device. Moreover, if attackers obtain K_n before the protocol is run, they are able to decrypt any message that is encrypted with K_n, as well as to derive K_{AB}. Another concern is replay of timestamps. In this protocol, protection against replay of earlier messages is achieved by validating the timestamps in the messages such that, if the timestamps in the message of an attacker differ from those in the legitimate messages, the attacker's message is rejected. However, an attacker can relatively simply forge the correct timestamp by observing when the messages in the legitimate protocol are transmitted and recording the time at that moment. Since the acceptance of timestamps at legitimate devices is adjusted to allow errors caused by currents, clock drift, encryption, and decryption delays, there is a possibility that the forged timestamp will be accepted as legitimate. Thus, stronger protection against replay attacks is needed. Generally, this can be provided by a counter, such as a nonce.

Limited forward secrecy After the establishment of K_{AB}, messages encrypted under it remain secure even if attackers possess K_n. However, this is only true if the attackers have not eavesdropped on or intercepted any previous messages. If previous protocol runs with messages encrypted with K_n are eavesdropped and stored, compromise of K_n at any point in time will give attackers the possibility to decrypt the messages and calculate previous session keys that depend on K_n. They can then decrypt all messages exchanged under the session keys as well. Hence, in this case, forward secrecy is not achieved.

Limited encryption strength RC5 was chosen because of its security provided by a long key and its relatively simple software implementation. However, Biryukov et al. [BK98] show that, with partial differential cryptanalysis, it is possible to derive all 25 subkeys for the 12 rounds of the 32-bit block version of RC5, with 2^{44} chosen plaintexts. It is not likely that this amount of messages will be encrypted under the same key in underwater communication, which makes such attacks infeasible. However, employing a stronger encryption algorithm would also eliminate even the theoretical possibility of such an attack. In addition, the usage of an IV is avoided due to the high packet loss in the underwater environment and the added bandwidth consumption necessary to transmit it. This implies that the cipher is used in ECB mode, which is generally regarded insecure. Especially since RC5 is used in the protocol with a block size of 32 bits, statistical attacks on each block may be possible.

Limited scalability The lookup table stored on each device requires substantial storage space and memory to store, read, and write to. Additionally, attempting to decrypt a received message with all long-term keys consumes large amounts of time and computational resources, resulting in poor performance, especially in resource-constrained devices, such as sensors. Especially if each long-term key is 2040 bits long, which is the maximum length allowed by RC5, the process of attempting decryption with each key, deriving the session key, and storing the session key in the correct location in the table is very long. In the worst case that the correct key is the last in the table, this method becomes infeasible. If large networks with many devices are to be supported by an authentication protocol of this kind, the scalability and performance must be improved, ideally decreasing the size of lookup tables and computational complexity to a minimum.

2.8.2 Encryption for Janus with Venilia

The recently published subclass of Janus, Venilia [HH21a], specifies an encryption scheme for Janus packets using the custom-made Tiny Underwater Block Cipher (TUBcipher) [HH21b]. Venilia is defined as Janus class 17, meaning that any packet with the Class user i.d. bits (bits 9 to 16 in Table 2.2) set to 17 (00010001_2) is

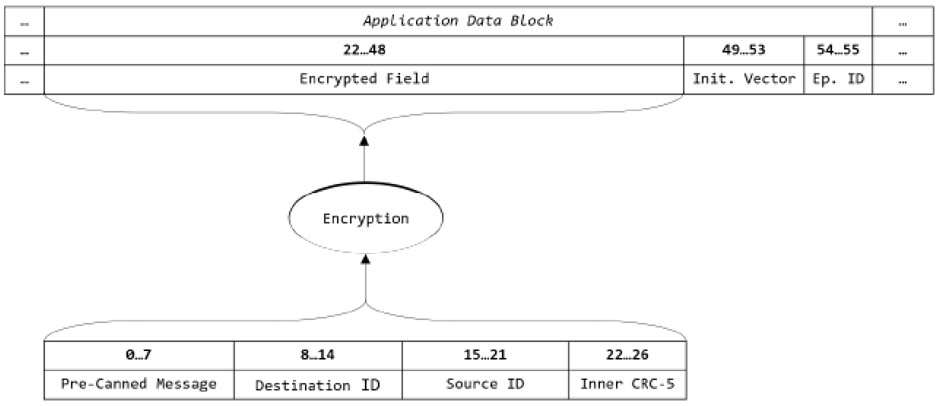


Figure 2.12: The encrypted portion of the ADB in the Venilia scheme. Source: [HH21a].

processed in the Venilia scheme. These packets also have the Schedule flag set to 0, as Venilia does not define cargo packets and channel reservation.

Venilia processing

27 of the 34 bits in the Janus ADB are encrypted. These 27 bits consist of an 8-bit message (so-called pre-canned message), a source address and destination address of 7 bits each, and an additional 5-bit Cyclic Redundancy Check (CRC). The remaining 7 bits in the ADB house a 5-bit IV and a 2-bit epoch identifier, both of which are used as input to the TUBcipher. The structure of the ADB can be seen in Figure 2.12.

The 8-bit message field allows for 256 unique messages to be processed. These messages must be stored in a pre-shared code book that is stored on all devices in a network. As each message has a specific, predefined meaning, a device must look up the message in the table upon reception and decryption.

The source and destination addresses are used to indicate the identities of the sender and receiver, respectively. The source address provides assurance that the message originated from a legitimate node in the network, and shows which node sent the message. The destination address indicates which node the message was designated for, meaning that the receiving node should reject a packet if the destination ID field does not match its own address. This usage of addresses implies that the total amount of 128 possible addresses must also be stored locally on each network node to allow for the validation of received source address fields. However, this is not explicitly stated in the Venilia specification.

The 5-bit inner CRC has the purpose of detecting decryption errors. It is thus

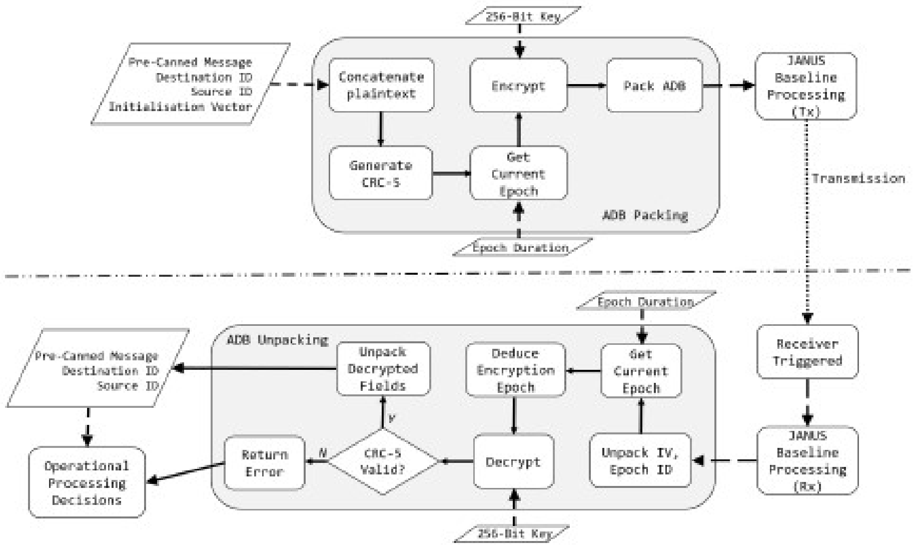


Figure 2.13: Transmission and reception processing of Venilia. Source: [HH21a].

solely meant for decryption verification by the receiver and it does not provide additional error correction for the Janus packet itself. It is calculated over the preceding 22 bits in the encrypted field with the polynomial:

$$p(x) = x^5 + x^4 + x^2 + 1. \tag{2.6}$$

The process of packing and encrypting a Janus ADB portion with Venilia, transmitting it with the Janus standard, and unpacking and decrypting the message is shown in Figure 2.13. The user is required to provide the message, node addresses, IV, key, and epoch duration. After concatenating the plaintext, generating the CRC, obtaining the current epoch, and encrypting, the resulting ciphertext is packed into the ADB. The ADB is then incorporated into a conventional Janus baseline packet, which is modulated and transmitted independently of Venilia. Upon reception, the receiving device also requires the epoch duration and key as input. The epoch is deduced by the epoch identifier transmitted in the packet. The ciphertext is then decrypted, before the CRC is validated. If the CRC is valid, the plaintext is obtained, otherwise, an error occurs.

TUBcipher

The TUBcipher performs the encryption and decryption operations in the Venilia scheme. It is defined in [HH21b]. A description of the operation of TUBcipher according to [HH21b] is given in Appendix A.

Limitations of Venilia

Although it would be desirable to use Venilia for our purposes and take the work of already established Janus security further, this system (Venilia and the TUBcipher) has certain limitations that make it unsuitable for our needs. This is especially the case if entity authentication is to be done with MMSIs. The limitations are outlined below.

Limited communication capabilities: A drawback of Venilia is that its operation is restricted to 8-bit C2 and status messages, which are stored in a predefined code book at each device in a communication network. Thus, there is a mere total of 256 messages that can be sent confidentially, which reduces the flexibility of communication. Additionally, the message code book would require the necessary storage space on each device, as well as a lookup mechanism to find the received message and make further operational processing decisions. This consumes time and energy. The fact that Venilia does not define the use of cargo packets further restricts its communication capabilities. Consequently, if 30-bit MMSIs are going to be used for entity authentication, Venilia cannot be used.

No authentication: Another problem is that it is assumed that authentication has already taken place before the use of Venilia and that all keys, code books, and clocks are synchronised locally before commencing communication. If authentication has not previously taken place, then there are many kinds of attacks that can be launched against the communication among devices, mostly using Man in the Middle (MitM) attacks. Therefore, there is a need to determine whether a potential counterpart in communication is a friend or foe. This can be achieved with an authentication scheme.

Short block size: The block size of 27 bits makes statistical frequency analysis by adversaries possible. The fact that the IV is only used for key derivation and not in the encryption process, implies that the TUBcipher is used in ECB mode. This makes statistical attacks an even more serious threat. We note that the key management of Venilia provides certain protection against such attacks. By including the epoch as input to the KDF, the linearity of the encryption is decreased, thereby reducing the success rate of frequency analysis attacks. Moreover, the length of 1 byte of every message that is transmitted is a vulnerability in itself due to ciphertext scarcity. If the aim of attackers is only to obtain the C2 message that is included in the 27-bit ciphertext, they can do so by trying a maximum of only 2^8 combinations, without performing any cryptanalysis.

2.9 Other Related Work

Most current research of wireless underwater security considers encryption and authentication schemes based on symmetric cryptography and pre-shared information, as this is the only model that is realizable with today's physical underwater infrastructure. Nevertheless, methods based on public key cryptography that would be applicable in a hypothetical underwater Public Key Infrastructure (PKI) have also been proposed.

ECDH and AES

An approach using public key cryptography is described in [GB18]. Here, key establishment between nodes in an underwater network is performed using the Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol [HL15]. After two nodes have established an identical key each, they then encrypt subsequent communications with a symmetric encryption algorithm, such as the AES. This scheme permits secure underwater machine-to-machine communication between the acoustic nodes in underwater networks. However, it assumes the existence of an underwater PKI, which is not specified. Also it is stated that some overhead may still be necessary in the form of message padding if a message is shorter than 128 bits, which is the block size used by AES. This implies that AES is used in either ECB mode or CBC mode, although the mode of operation itself is not specified. Due to the low bandwidth requirements of Janus, the need of this amount of padding may add a significant delay to the transfer of messages.

Physical Layer Security

An authentication method that does not use cryptographic techniques, but rather the statistical features of the underwater communication channel, is presented by Diamant et al. [DCT19]. This method is based on cooperative authentication, meaning that several network nodes are involved in determining the authenticity of a received packet. More specifically, a controller node (also called a sink) authenticates a packet from a sender node with the help of several trusted nodes. The feature distribution of the channel is derived from the received packets at the trusted nodes and the sink uses this distribution to distinguish packets sent from legitimate nodes and attackers. However, schemes based solely on the properties of the physical communication channel may not be secure enough and, therefore, classical cryptographic solutions may be needed in this environment as well.

Custom Cipher

An encryption algorithm based on a block cipher for general UANs is described by Peng et al. [PDLL16]. The algorithm utilizes 64-bit blocks, 8 rounds of encryption, and 32-bit subkeys for each round. It makes use of a logistic map to provide substitution and thereby reduces the computational cost of encryption compared to the cost of a traditional S-box. Generally, it only employs simple operations, such as XOR and bit shifting, resulting in increased speed and reduced storage space, but also security implications due to a smaller key size than other standardized algorithms. In addition, the algorithm only provides confidentiality of messages, without details of an authentication scheme.

Chapter 3

Authenticated Encryption Schemes for Janus

In this chapter, we present the AEAD schemes we have identified as potentially suitable for usage with Janus, namely CCM and AEGIS. One of the contributions of this thesis, which we expose here, is to specify the adaptations of their original descriptions for use with Janus, both with and without cargo packets. We give argumentation for their suitability in this environment, based on their efficiency and security properties. Specifically, both algorithms provide the necessary security requirements outlined in Section 2.5 with little computational and communication overhead. We also define enhancements of the authentication protocol by Téglásy et al. [TWPK], described in Section 2.8.1, using both algorithms.

3.1 Authenticated Encryption with CCM

Given the security and bandwidth requirements, a potentially suitable algorithm for providing assurance of the confidentiality and the integrity of Janus data is the CCM mode of operation for block ciphers, proposed by Whiting et al. [WHF02] and standardized as Special Publication 38-C of the NIST [Dwo04]. The CTR mode it applies for encryption allows for the avoidance of transmitting complete blocks of the underlying block cipher. Even though the data must be padded for partitioning into complete blocks, this happens locally on a device, and complete blocks do not need to be transmitted. The only additional overhead in the data that is sent are the nonce and the MAC tag. It is therefore a good option for our needs as it is a standard that provides a high level of security, and the Janus baseline packet and cargo can be formatted as input to the CBC-MAC and encryption algorithms.

According to [Dwo04], CCM is only defined for block ciphers with a block size of 128 bits. Currently, the only approved cipher by NIST is AES. There are three inputs to the CCM algorithm: the plaintext P (also called the payload) to be authenticated and encrypted, the associated data AD that will only be authenticated and not encrypted, and a nonce N, which is a unique value associated to the other two inputs. Typically, the payload consists of user data that needs to be confidential and authentic, while the associated data are packet headers that are also authentic, but remain not encrypted for the proper functioning of routing mechanisms in networks. In our case, the payload consists of the ADB in the Janus baseline packet and optional cargo. If cargo is not used, the associated data is the 22-bit preamble before the ADB, and the two 1-bit flags. If cargo is used, the first byte of the ADB is added to the associated data. The nonce can be a random number or any value that is unique under a given key. CCM defines two major operations: generation-encryption, in which the MAC tag is generated and the payload and the MAC tag are encrypted, and decryption-verification, where the payload and MAC tag are decrypted and the MAC tag is verified.

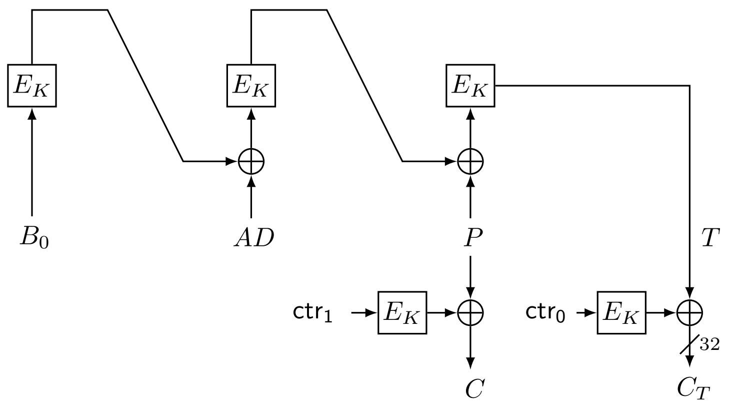
We consider the example where only the 64-bit baseline Janus packet is to be processed by CCM. The length of the secret key K_n will be at least 128 bits, as described in [TWPK]. The MAC tag T should, by the specification, have a length of at least 4 bytes (32 bits) to prevent forgery attacks. The octet length of N, n, should, by the specification, be at least 7. The total length of AD is 30 bits, while the length of P is 34 bits.

Bearing in mind the discussion above, the variables defined by CCM for use in Janus that we have chosen are the following:

- K_{len}: the key length in bits, in our case at least 128.
- T_{len}: the length of the tag in bits, in our case 32.
- N_{len} : the length of the nonce in bits, in our case 64 (see the explanation in Section 3.1.5).
- AD_{len}: the length of the associated data in bits, in our case 24.
- P_{len}: the length of the payload in bits, in our case 34 (per packet).
- t: the length of the tag in bytes, in our case 4.
- n: the length of the nonce in bytes, in our case 8.
- a: the length of the associated data in bytes, in our case 3.
- p: the length of the payload in bytes, in our case 5 (P_{len} padded with 6 bits).
- q: the length in bytes of Q, which is the bit string representation of p. Calculated as $15 - n$, in our case $15 - 8 = 7$.

If cargo is used, AD_{len} will increase to 32 bits in each packet, as the first byte of the ADB will specify the cargo properties and thereby be regarded as meta-information.

Below, we give the details of the CCM algorithm as described in [Dwo04], with the use of the variable values specified above. A general overview of the CCM mode of operation for a Janus baseline packet is shown in Figure 3.1.



$B_0 = \mathsf{Flags_1}||N||\mathsf{Q}$

$\mathsf{ctr_i} = \mathsf{Flags_2}||N||i$

Figure 3.1: Overview of CCM used to process a Janus baseline packet. AD must be formatted for CBC-MAC, and P must be formatted for both CBC-MAC and CTR encryption. The final ciphertext is $C||C_T$. Adapted from [Jea16].

Table 3.1: Formatting of block B_0 in CCM. Source: [Dwo04].

Octet number	0	$1...15-q$	$16-q...15$
Contents	Flags	N	Q

3.1.1 Generation of the MAC Tag

Before the MAC tag T can be generated, N, AD, and P need to be formatted such that they can be represented as 16-byte blocks $B_0, B_1, ..., B_r$, which will be the input to the CBC-MAC algorithm.

Formatting of the Control Information and the Nonce

The control information and the nonce N are represented in the first block, B_0. B_0 consists of a flag octet, followed by N, followed by Q. The formatting of B_0 is shown in Table 3.1.

The flag octet of B_0 is formatted as shown in Table 3.2. Here, t is the octet length of the MAC tag T and q is the octet length of Q. Bits 5, 4, and 3 consist of the 3-bit representation of the value $(t-2)/2$. If we set $t=4$, which is our choice for Janus, this value will be $(4-2)/2 = 1 = 001_2$. Similarly, bits 2, 1, and 0 are the 3-bit representation of $q-1$. If $q=7$ (our Janus case), then their value will be $7-1=6=110_2$. The *Reserved* bit should by definition be set to 0. The *Adata* bit shall be set to 1 if there is associated data to authenticate, and 0 otherwise. Since

Table 3.2: Formatting of the flag octet of block B_0 in CCM. Source: [Dwo04].

Bit number	7	6	5	4	3	2	1	0
Contents	Reserved	Adata	$[(t-2)/2]_3$			$[q-1]_3$		

there usually is such data with Janus communication, it will be set to 1. With the values given, the flag octet will be 01001110_2.

Following the flag octet, the octets $1...15-q = 1...15-7 = 1...8$ are taken up by N.

The last 7 bytes of B_0 consist of Q, whose value is $[p]_{8q}$. With $p = 5$ and $q = 7$, $Q = 00000000000000000000000000000000000000000000000000000101_2 = 5_{10}$. The large amount of leading 0-bits is necessary to fill the last portion of block B_0. This has no impact on the security that is provided and the computational overhead is negligible.

Formatting of the Associated Data

B_0 is followed by the associated data, which is partitioned into the blocks $B_1, B_2, ..., B_u$, where u is an integer that depends on the size of AD. AD is encoded depending on its octet length, a. According to [Dwo04], if $0 < a < 2^{16} - 2^8$, a is encoded as $[a]_{16}$, i.e., two bytes. In the case of Janus without cargo, $AD_{len} = 24$, hence, 3 bytes are needed to represent it with a. Thus, a is encoded with two bytes as 0000000000000011_2. This string is concatenated with the actual 24-bit value of AD and the resulting string is padded with 0-bits such that the string can be partitioned into the 16-byte blocks $B_1, B_2, ...B_u$. In our case, $u = 1$, and the associated data can be represented in a single block, B_1.

Formatting of the Payload

The payload follows after the associated data blocks, and it is partitioned into the blocks $B_{u+1}, B_{u+2}, ..., B_r$, where $r = u + \lceil \frac{p}{16} \rceil = 1 + \lceil \frac{5}{16} \rceil = 1 + 1 = 2$. Hence, the payload fits into one block, B_2, with its 5 bytes padded with 0-bits to a length of 16 bytes.

Application of CBC to Generate the MAC Tag

After the formatting procedure, the blocks B_0, B_1, and B_2 are input to the CBC-MAC algorithm. The algorithm employs AES in CBC mode over the formatted blocks to produce the MAC tag T of length T_{len}, which is chosen in advance. In our case it is 32 bits. The algorithm is as follows:

1. Set the initial block $Y_0 = AES_{K_n}(B_0)$.

Table 3.3: Formatting of the counter blocks Ctr_i in CCM. Source: [Dwo04].

Octet number	0	$1...15-q$	$16-q...15$
Contents	Flags	N	$[i]_{8q}$

Table 3.4: Formatting of the flag octet of counter block Ctr_i in CCM. Source: [Dwo04].

Bit number	7	6	5	4	3	2	1	0
Contents	Reserved	Reserved	0	0	0	$[q-1]_3$		

2. For $i = 1$ to r, do $Y_i = AES_{K_n}(B_i \oplus Y_{i-1})$.
3. Set $T = MSB_{T_{len}}(Y_r) = MSB_{32}(Y_r)$.

After the execution of the algorithm, T consists of the 32 Most Significant Bits (MSBs) of the last block in the CBC process, Y_r.

3.1.2 Encryption

The encryption in CTR mode relies on a counter generation function, which generates the 16-byte counter blocks $Ctr_0, Ctr_1, ..., Ctr_m$ to be encrypted. Like the blocks used as input to the CBC-MAC algorithm, the counter blocks also require formatting.

Formatting of the Counter Blocks

Each counter block, Ctr_i is formatted as shown in Table 3.3. The values of N and q are the same as for the MAC tag generation. In the last portion of a block, the counter i is represented as q octets.

The formatting of the Flags octet is as shown in Table 3.4. This octet is identical in all blocks Ctr_i. The Reserved bits shall by the specification be set to 0 and the value $[q-1]_3$ is the same as the one derived during the formatting of the Flags octet in block B_0.

Application of CTR to Produce the Ciphertext

The number of counter blocks is determined as $m = \lceil \frac{P_{len}}{128} \rceil$. Since P is the 34-bit Janus ADB, $m = \lceil 0.27 \rceil = 1$. Thus, there will be two counter blocks used for encrypting P and T, Ctr_0 and Ctr_1. The encryption algorithm in concrete Janus form is as follows:

1. For $j = 0$ to m, do $S_j = AES_{K_n}(Ctr_j)$.
2. Set $S = S_1 || S_2 || ... || S_m = S_1$.
3. Return $C = (P \oplus MSB_{P_{len}}(S)) || (T \oplus MSB_{T_{len}}(S_0)) = (P \oplus MSB_{34}(S_1)) || (T \oplus MSB_{32}(S_0))$.

The counter blocks Ctr_j are encrypted to produce the keystream blocks S_j. There will again be two such blocks, S_0 and S_1, both 16 bytes long. The MAC tag is encrypted by applying the XOR operation on the 32 MSBs of S_0 and T, and the payload is encrypted by the same operation on the 34 MSBs of S_1 and P. The resulting two strings are concatenated to produce the final ciphertext C.

3.1.3 Choice of Length of the MAC Tag for Janus

Appendix A in NIST SP 800-38B [Dwo05] provides guidance for the choice of length of the authentication tag in the CBC-MAC algorithm. It is stated that if T_{len} is chosen to be less than 64, careful analysis should be done of the risks implied by accepting an inauthentic message and that the number of tag verification failures before retiring the key should be restricted sufficiently. The formula given for the derivation of the tag length is shown below.

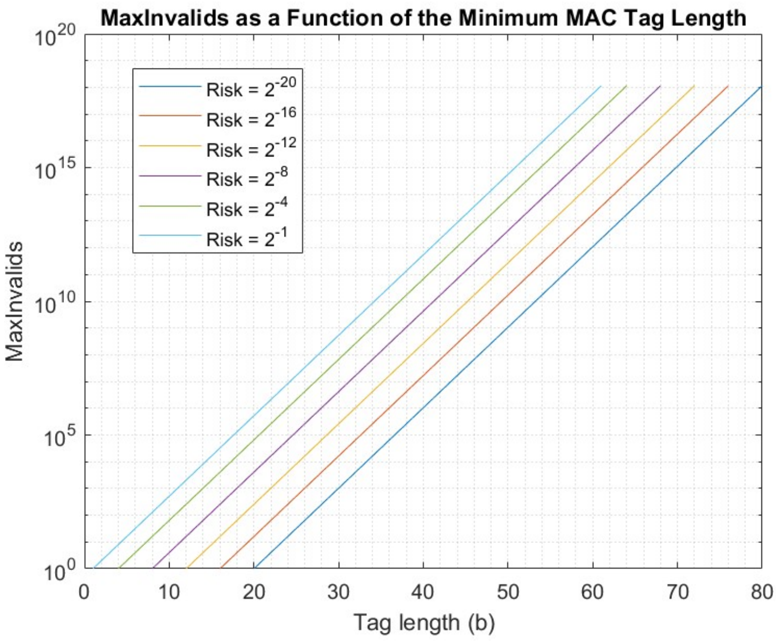
$$T_{len} \geq \log_2 \left(\frac{MaxInvalids}{Risk} \right) \tag{3.1}$$

In the formula (3.1), $MaxInvalids$ refers to the maximum allowed number of verification failures of the MAC tag before retiring the key for generation and verification, while $Risk$ is the highest acceptable probability for an inauthentic message to pass the verification process.

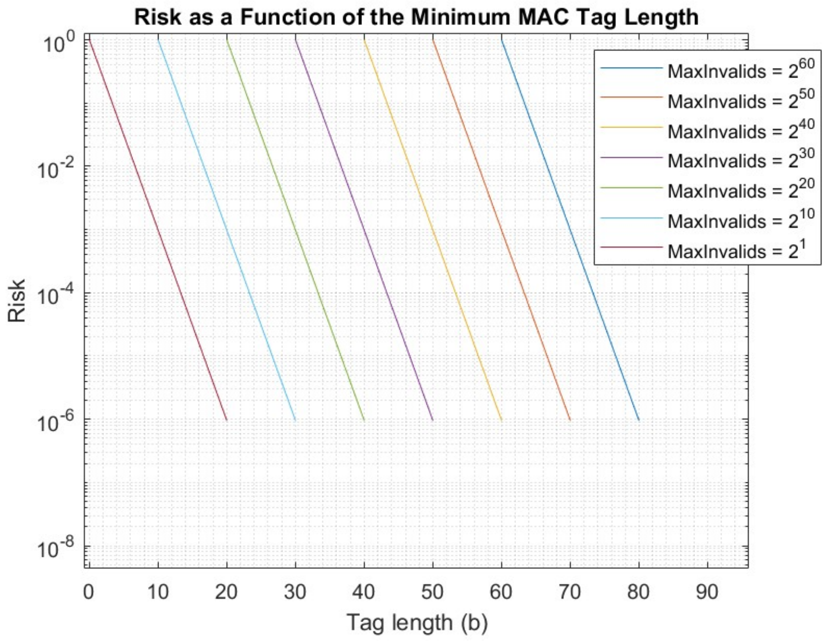
Following this guidance, a comparison of values of T_{len} with varying values of $MaxInvalids$ and $Risk$ is shown in Figure 3.2. We observe that a longer tag provokes a lower $Risk$ value and vice versa, while a shorter tag gives the need to reduce the value of $MaxInvalids$ due to the higher probability of successfully forging a valid tag. If we choose a desirable value of $Risk$ to be 2^{-12} with $T_{len} = 32$, the corresponding value of $MaxInvalids$ is 2^{20}. In other words, we allow for 2^{20} unsuccessful forgery attempts before retiring the key. Since the limited communication capabilities of Janus make it infeasible to launch this number of forgery attempts under the same key, we conclude that a tag length of 32 bits with a limit of 2^{20} authentication failures is an acceptable combination.

3.1.4 Argumentation for the Use of CCM with Janus

We decided to use CCM first and foremost due to its familiarity as a mode of operation for AEAD and its long-lasting and widespread use in Wireless Local Area Networks (WLANs). Despite the age of the algorithm, very few practical attacks against both its confidentiality and authenticity have been developed, meaning that it still provides a high level of security. Another argument for its usage in underwater



(a) *MaxInvalids* as a function of the MAC tag length, with varying values of *Risk*.



(b) *Risk* as a function of the MAC tag length, with varying values of *MaxInvalids*.

Figure 3.2: Comparisons of minimum MAC tag length for the CBC-MAC algorithm.

networks is its ability to partially encrypt a message, while authenticating both the encrypted and unencrypted parts. Since the Janus packets in our authentication protocol have messages of this form (i.e., it is possible to format the Janus header as associated data, while only encrypting the ADB), formatting our messages for the structure of CCM is relatively simple. Following are details of the general usage of CCM, and its security properties.

Historical Context and Alternative Algorithms

CCM was standardized by NIST in 2003 and it is still widely used today. It became part of the 800-38 series of NIST special publications after it was submitted in a public contest for block cipher modes of operation. It was selected because it served the important need to improve the encryption, data integrity, and header integrity algorithms in IEEE 802.11 WiFi networks. The algorithms it replaced were RC4 for encryption and Michael [Fer02] for integrity, used in the previous iteration of the WiFi security architecture, WiFi Protected Access (WPA). The next iteration, called WPA2, instead used CCM, as well as the Robust Security Network (RSN) protocol for establishing secure communication over IEEE 802.11 networks. WPA2 became the final draft of the IEEE 802.11i standard, an overview of which is given by, for instance, Chaplin et al. [CQP+05]. In addition to CCM (known as CCM Protocol (CCMP) in the context of WiFi), two other data confidentiality algorithms are defined in IEEE 802.11i, namely Wired Equivalent Privacy (WEP) and Temporal Key Integrity Protocol (TKIP). These were used in older systems and were also replaced by CCM. Naturally, CCM provides the strongest confidentiality of the three, as numerous vulnerabilities have been found in the older algorithms. A security analysis of IEEE 802.11i, including an analysis of these algorithms is given by He and Mitchell [HM05].

The major competitor to CCM is Galois/Counter Mode (GCM), which is also standardized by NIST as SP 800-38D [Dwo07]. Like CCM, it is also a mode of operation for AES that provides AEAD, and it is also used in WPA2. It realizes confidentiality by encrypting a message with AES in CTR mode, where the initial counter block is created by incrementing a block generated from the IV. Subsequent counter values are obtained by incrementing the counter value modulo 32. The authentication, unlike CCM, is achieved with a keyed hash function, called GHASH. It utilizes multiplication by a fixed subkey, which is generated by encrypting the first block with AES. The multiplication takes place in a Galois Field $GF(2^{128})$. GHASH is used to compress an encoding of the associated data and the ciphertext to produce an authentication tag.

GCM has certain advantages over CCM. For instance, CCM cannot be processed in a pipeline or in parallel because CBC-MAC cannot be executed in parallel processes.

For this reason, GCM, especially with software pre-computation, where time is traded for memory, is generally faster than CCM. Another advantage of GCM is that it can be used online, i.e., messages can be processed in real time while they are generated or transmitted. On the contrary, CCM requires that the whole message to be processed is known in advance. An advantage of CCM is that it is very easy to implement, both in hardware and software, as it does not involve polynomial multiplications. The multiplication over $GF(2^{128})$ in GCM is more challenging to implement and requires additional hardware or software pre-computation. The software pre-computation can also reduce the performance of GCM if used too frequently, as it must be done for each subkey in GHASH. A performance comparison of different versions of CCM and GCM is given by Szalachowski et al. [SKK10], where the standard version of CCM is shown to consume less memory and perform initialization in fewer cycles than the equivalent version of GCM.

In terms of security, CCM is provably secure, as discussed below, while GCM, despite also having a security proof [MV04], is shown by Ferguson [Fer05] to have weaknesses. For instance, the IV in GCM is required to be at least 96 bits long, otherwise, IV collisions are possible. This can be inconvenient, since such a long IV can be expensive for small devices, and it can increase bandwidth consumption, which is a crucial drawback in our underwater applications. Ferguson also shows that GCM provides less security of authentication than expected, when a short authentication tag is used. Again, since our aim is to use tags that are as short as possible, CCM provides better assurance against forgery attacks. Finally, key forgery attacks are also possible against GCM due to the linear behavior of the authentication function. It relies on multiplications by a constant and squaring, which are both linear operations. For this reason, nonce reuse attacks (to which both CCM and GCM are vulnerable) are much more effective against GCM, since the attacks often lead to the possibility of obtaining the authentication key. Vanhoef and Piessens [VP17] demonstrate the effects of nonce reuse against both algorithms in the context of IEEE 802.11 authentication.

General Properties

Jonsson [Jon03] provides a formal analysis of CCM with the conclusion that a high level of confidentiality and authenticity is provided in line with other standardised modes of operation for authenticated encryption, such as GCM or Offset Code Book Mode (OCB). The following attractive properties of CCM are listed:

1. A specific mechanism that handles parts of messages that are only to be authenticated and not encrypted is provided by default, which is very convenient for use with Janus. This is done without additional ciphertext overhead, something that requires enhancements in many other authenticated encryption modes.

2. AES is used only in the forward direction for encryption and its inverse is never used. This is the case in both the generation-encryption and decryption-verification operations. This contributes to the reduction of the code size of implementations of CCM, which improves the efficiency of implementation in Janus.
3. The CTR and CBC-MAC algorithms are widely deployed and have been in use for a long time, meaning that CCM is based on well known and established technology. Comprehensive documentation of the algorithms is widely available, which helps to avoid implementation faults. Existing implementations are also highly optimized.
4. All intellectual property rights have been released to the public domain, making CCM freely available for any purpose.

Authenticity Analysis

Regarding authenticity, Jonsson [Jon03] states that it is hard to extract any nontrivial information about the input blocks B_i and the output blocks Y_i of the CBC-MAC algorithm, even if all the plaintexts of a message exchange are known. Because of this, if an attacker modifies any previous encrypted messages, the effects on the encrypted MAC tag will be unpredictable. Thus, any attempt to forge the tag must utilize a unique sequence of blocks $B_0, B_1, ...B_r$. This means that it is hard for an attacker to guess a valid MAC tag with a probability higher than $2^{-T_{len}}$.

For an encryption query $Q = (N, AD, P)$, the number of applications of the underlying encryption algorithm (in our case AES), l_Q, is expressed as

$$l_Q = \left\lceil \frac{|f(N, AD, P)| + |P|}{128} \right\rceil + 1, \tag{3.2}$$

where f is the formatting function used to generate the input blocks $B_0, B_1, ...B_r$.

For a forgery attempt with an encrypted message C^*, $Q = (N^*, AD^*, C^*)$, the number of applications of AES to decrypt C^* and check whether C^* is valid is given as

$$l_{Q^*} = \left\lceil \frac{|f(N^*, AD^*, P^*)| + |C^*|}{128} \right\rceil + 1. \tag{3.3}$$

The sum of AES applications for all encryption queries and all forgery attempts is then given as

$$l_E = \sum_i l_{Q_i} \qquad \text{and} \qquad l_F = \sum_i l_{Q_i^*}, \tag{3.4}$$

respectively.

To successfully forge a MAC tag, the goal of an attacker A is to distinguish the output of the CBC-MAC algorithm from that of a Pseudo-Random Function (PRF) and, since AES is a block cipher, a Pseudo-Random Permutation (PRP). For this, distinguishers B and B' are used, respectively. Regarding the indistinguishability from a PRF, the advantage of A over the authenticity of CCM in terms of the PRF is upper-bounded by

$$Adv_{CCM}^{auth}(A) \leq Adv_{AES}^{prf}(B) + q_F \cdot 2^{-T_{len}} + (l_E + l_F)^2 \cdot 2^{-128-1}, \tag{3.5}$$

where q_F is the number of forgery attempts.

Similarly, the upper bound of the advantage of A in terms of the PRP is given as

$$Adv_{CCM}^{auth}(A) \leq Adv_{AES}^{prp}(B') + q_F \cdot 2^{-T_{len}} + (l_E + l_F)^2 \cdot 2^{-128}. \tag{3.6}$$

Since the attacker can only guess valid input blocks, the most effective attacks against authenticity are based on finding collisions among forgery attempts and valid tags. Thus, the security proof is related to the birthday paradox, which states that collisions $X_i = X_j$ are likely to be found among l random strings $X_1, ..., X_l$ of length 128 (the block size of AES) if l is approximately $2^{\frac{128}{2}}$. This can also be seen from the term of the form $c \cdot l^2 \cdot 2^{-128}$, where c is a constant.

The analysis presented above is from [Jon03] and is very general. In our particular case with Janus, the only specific factor that could influence the security of authentication is the length of the authentication tag T_{len}. We have already explained previously (Section 3.1.3 and Figure 3.2) that the specific conditions that hold in underwater communication (low bit rate) prevent exploiting the relatively short MAC tag that we use with Janus. Therefore, there is no reason to believe that the security of the CCM authentication procedure can be threatened in the Janus-based implementation.

Confidentiality Analysis

With respect to confidentiality, the goal of an attacker would again be to distinguish a ciphertext from a bit string chosen uniformly at random. The two given ways that this can be done is either that the attacker executes a birthday attack against the CTR output blocks or that an anomaly occurs within the CBC-MAC computation, such as an internal collision or a MAC tag that is identical to some CTR output block. The lack of other possible attacks is due to the fact that N is required to be fresh during the lifetime of a key, hence, the CTR input blocks Ctr_i and the first block B_0 of CBC-MAC are new. This will make a ciphertext strongly resemble a random string, even if the attacker knows the plaintext.

The advantage of the adversary A over the privacy of CCM is, as for the authenticity, upper-bounded by the advantage of a distinguisher B over the PRF of AES and a distinguisher B' over the PRP. For B, the bound is

$$Adv_{CCM}^{priv}(A) \leq Adv_{AES}^{prf}(B) + l_E^2 \cdot 2^{-128-1}, \tag{3.7}$$

where l_E is as defined above.

Similarly, for B', the bound is

$$Adv_{CCM}^{priv}(A) \leq Adv_{AES}^{prp}(B') + l_E^2 \cdot 2^{-128}. \tag{3.8}$$

Again, as for the authenticity, the security proof is limited by the birthday paradox, forcing the attacker to find collisions between valid ciphertext and pseudo-random text.

The application in Janus does not change any parameters of CCM related to confidentiality compared to the general definition. Therefore, the analysis above applies directly to our Janus application without any changes. Thus, the confidentiality part of CCM in Janus can be considered secure as long as the implementation is correct.

3.1.5 Application in Janus-based Communication

To perform both authentication of messages and entity authentication, regardless of the security algorithm that is used, it is necessary to transmit the following values:

1. A 29-bit timestamp TS
2. A 3-bit clock accuracy descriptor CD
3. Two 1-bit SYN and ACK flags F
4. The MMSIs of both device A and B

These values must be sent by both devices, in each direction. The transmission of both MMSIs allows the sending device to authenticate itself and to indicate which device the message is designated for. Thus, the lookup table in the original version by Téglásy et al. (Table 2.4) is reduced, as it is only necessary to store the MMSIs of all devices in it, to allow for the verification of the received MMSIs. There is no longer a need to store a long-term key K_n and session key K_{AB} for each device. This improves scalability, as the storage of the original lookup tables and the lookup procedures would require much storage space and processing as the network grows. For CCM, the required input elements at the receiving device are the nonce N, the associated data AD, and the ciphertext C, which consists of the plaintext P and MAC tag T, both encrypted. Hence, our goal is to partition the values used in the authentication protocol into the format of CCM.

Regarding the application in Janus, CCM can be directly applied, using the smallest recommended values of t, n, a, and p, such that the standardized levels of confidentiality and authenticity are provided. However, the output of the processing of a Janus baseline packet cannot itself fit into a single baseline packet due to the need to transmit the nonce N and the MAC tag T in addition to the associated data AD and the encrypted payload C. Instead, either a cargo must be specified or several baseline packets must be used. Which option is better depends on several factors, such as the scale of the network and the amount of data that the devices send on average. Since the aim of Janus is to be an open standard, we believe that users with different requirements may wish to use either option. Following are descriptions of the incorporation of CCM into the protocol for authentication of underwater assets defined by Téglásy et al. [TWPK] and described in Section 2.8.1, using both options.

Without Usage of Cargo

When cargo packets are not used, the values listed above, as well as the nonce and authentication tag must be partitioned into five baseline packets. Below we propose how this can be performed.

The Nonce N For CCM to function, N must have a length of at least 56 bits and cannot fit into the 34-bit ADB entirely. However, a way to resolve this is to set an initial length of N to 32 bits and then duplicate that value locally at each device to produce a 64-bit string, which is a valid length for the generation-encryption and decryption-verification operations. The entropy of N will in this case still be the same as for a 32-bit string. However, a 32-bit string allows for N to be unique for 2^{32} messages under the same key, which is a satisfactory amount for most applications.

The Associated Data AD Regarding the associated data, we assume that the CRC does not need to be authenticated, as its sole purpose is to detect errors in transmission. Thus, the consequences of accepting an inauthentic CRC are not grave. On the other hand, F should be authenticated, as it provides relevant status information in the protocol. Taking this into account, AD consists of the 22-bit

Janus header (as shown in Table 2.2) and the two 1-bit flags, so it is 24 bits long. In our case, the Janus header will remain the same for all CCM-related packets sent in one direction that are part of the authentication protocol. This allows for the usage of the Janus header of any packet to directly be included in AD, without the need to fit AD in the ADB. Thus, the first baseline packet can be used to transport both N and the 22 bits of AD that are contained in the Janus header. The two remaining bits of AD (F) can either be transmitted in the same packet or in the next one. In Figure 3.3, they are transmitted in the second packet.

The Timestamp TS and Clock Accuracy Descriptor CD The timestamp and clock accuracy descriptor are used for the ranging functionality. These values consist of 32 bits in total and can therefore fit in the ADB together with the flags F. TS and CD can also directly be encrypted in the CTR mode of CCM such that a 32-bit ciphertext is produced. Hence, the second packet in the protocol is used to transport TS and CD in encrypted form and F in plaintext.

The MMSIs The MMSIs have a length of 30 bits each and must be placed in their respective baseline packets. Therefore, the third and fourth packets are used to transport $MMSI_A$ and $MMSI_B$.

The MAC tag T The authentication tag T is the final element that needs to be transmitted. T is generated from N, AD, and P at the sending device from the input blocks B_0, B_1, and B_2, before it is encrypted together with P. Since there is an entire baseline packet into which to place T, it is possible to set the size of T to 4 bytes and conveniently put it in the ADB.

The Final Packet With the transmission of T in the fifth packet, the transmission of the necessary values for both authentication of messages and entity authentication, authenticated and encrypted with CCM, is completed. The receiving device will repeat the process with its own timestamp, clock accuracy descriptor, combination of flags, and the MMSIs for its response. The complete protocol can be seen in Figure 3.3. As each participant transmits five 64-bit baseline packets, the theoretical delay will be $64 \cdot 5/80 = 5s$, in addition to the propagation delay of the acoustic signals, which depends on the distance between the participants.

With Usage of Cargo

The usage of a cargo would allow for transmitting all the required data in a single packet, thus avoiding the need to transmit AD and a CRC for every baseline packet, which leads to reduced bandwidth consumption. However, the channel would be reserved for the transmission of the cargo, preventing any other devices from transmitting while the cargo is being transmitted.

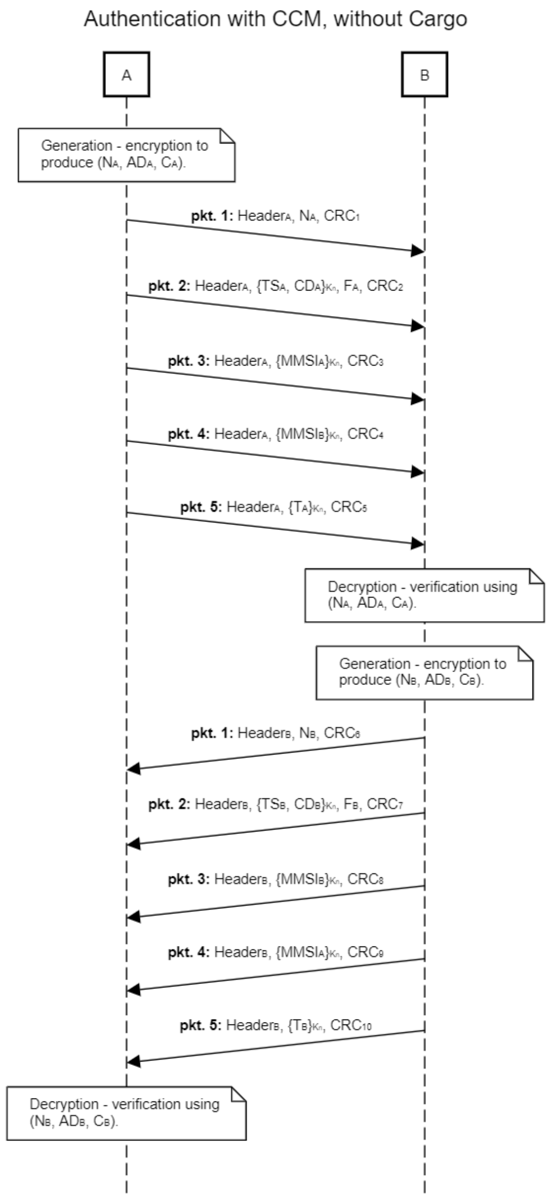

Authentication with CCM, without Cargo
A
B
Generation - encryption to produce (N_A, AD_A, C_A).
pkt. 1: $Header_A$, N_A, CRC_1
pkt. 2: $Header_A$, $\{TS_A, CD_A\}_{K_n}$, F_A, CRC_2
pkt. 3: $Header_A$, $\{MMSI_A\}_{K_n}$, CRC_3
pkt. 4: $Header_A$, $\{MMSI_B\}_{K_n}$, CRC_4
pkt. 5: $Header_A$, $\{T_A\}_{K_n}$, CRC_5
Decryption - verification using (N_A, AD_A, C_A).
Generation - encryption to produce (N_B, AD_B, C_B).
pkt. 1: $Header_B$, N_B, CRC_6
pkt. 2: $Header_B$, $\{TS_B, CD_B\}_{K_n}$, F_B, CRC_7
pkt. 3: $Header_B$, $\{MMSI_B\}_{K_n}$, CRC_8
pkt. 4: $Header_B$, $\{MMSI_A\}_{K_n}$, CRC_9
pkt. 5: $Header_B$, $\{T_B\}_{K_n}$, CRC_{10}
Decryption - verification using (N_B, AD_B, C_B).

Figure 3.3: CCM used in the authentication protocol, without the usage of cargo packets.

The specification of a cargo is done by a leading baseline packet with the Schedule flag set to 1 and an appropriate amount of time reservation specified in the ADB. These fields are shown in Table 2.2. AD again consists of the 22-bit Janus header and F, but, additionally, the first byte of the ADB is considered as associated data, as it provides meta-information about the packet. Since the first byte of the ADB is used for the cargo specification, 26 bits are left in it for user data. Thus the 26 MSBs of N can be placed here, while the 6 Least Significant Bits (LSBs) are transmitted as cargo. Following N, the encrypted payload is transmitted, consisting of the ciphertexts of TS, CD, the two $MMSIs$, and T. The protocol can be seen in Figure 3.4, while the packet format is shown in Figure 3.5.

For the case of one packet of this format, the total amount of data is $30 + 26 + 8 + 6 + 2 + 29 + 3 + 30 + 30 + 32 = 196$ bits, with the cargo consisting of the last 132 bits. With the data rate of 80 bps provided by Janus, the theoretical encoding time for the entire packet is $196/80 = 2.45$ s. For the cargo alone, the encoding time is $132/80 = 1.65$ s. This means that at least 1.65 s of channel reservation time must be specified in the first byte of the ADB. According to the lookup table for channel reservation used by Janus, the minimum time that can be reserved with the reservation bits in the ADB to accommodate this cargo length is 1.79142724 s, corresponding to index 66 in the table. The bits are thus set to 1000010. Compared to the encoding time for one baseline packet alone, $64/80 = 0.8$ s, this packet format does not impose a long additional delay.

Summary of the CCM-Based Scheme

Regardless of the increased amount of data that is transmitted, CCM offers much more flexibility in terms of the kind and amount of data that can be transmitted securely in comparison to, for instance, the Venilia scheme, which offers confidentiality only for one predefined 8-bit message at a time and no authenticity.

In the previous sections, both cargo and cargo-less solutions are provided. As mentioned previously, each version has advantages and disadvantages. The version without cargo requires the transmission of more data due to repeated need to include the Janus header and CRC. This leads to higher Round-Trip Times (RTTs) of the protocol due to the need to encode and transmit more data. Additionally, this version is more vulnerable to packet loss. On the other hand, when cargo is used, the RTT decreases significantly, as well as the power consumption. However, when cargo is not used, there is no need to reserve the channel at any point during the protocol. This can be beneficial if other devices in the network are transmitting mission-critical information which must have minimal delays. When using cargo, other devices would have to wait until the cargo is transmitted until they can transmit themselves. It is worth noting that the very short amount of channel reservation time that is needed

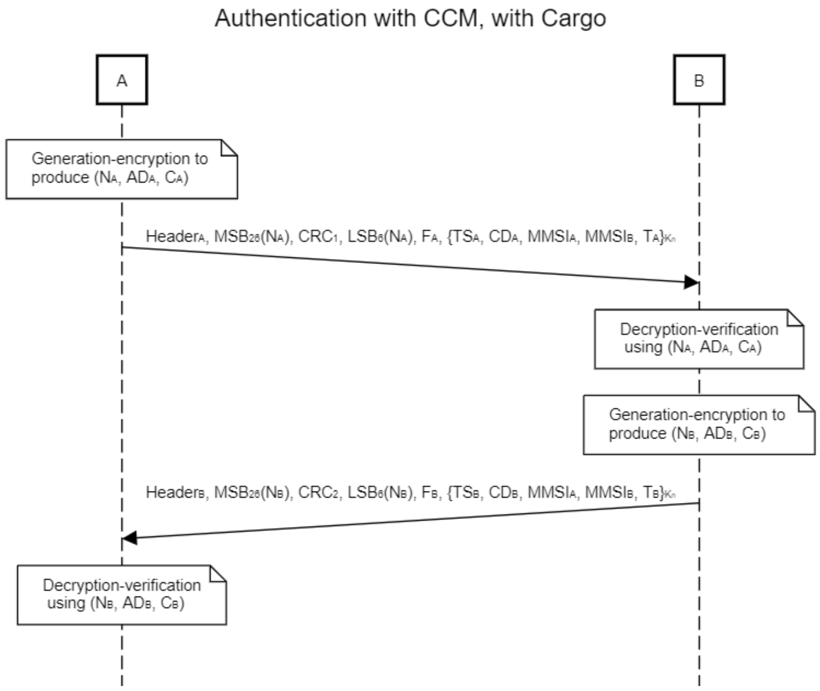

Authentication with CCM, with Cargo
A
B
Generation-encryption to produce (NA, ADA, CA)
HeaderA, MSB26(NA), CRC1, LSB6(NA), FA, {TSA, CDA, MMSIA, MMSIB, TA}Kn
Decryption-verification using (NA, ADA, CA)
Generation-encryption to produce (NB, ADB, CB)
HeaderB, MSB26(NB), CRC2, LSB6(NB), FB, {TSB, CDB, MMSIA, MMSIB, TB}Kn
Decryption-verification using (NB, ADB, CB)

Figure 3.4: CCM used in the authentication protocol, with the usage of cargo packets.

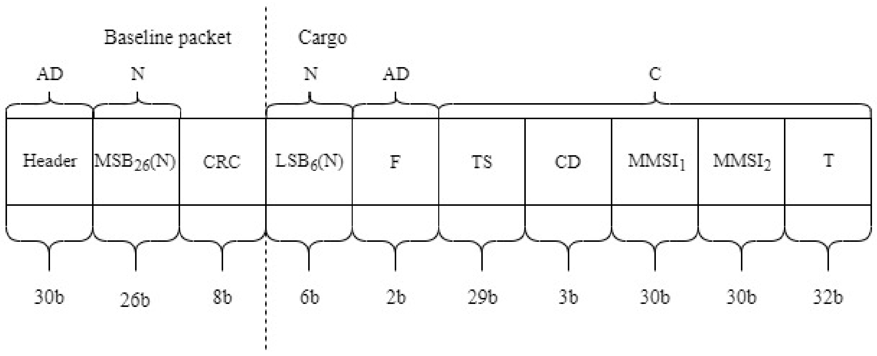


Figure 3.5: Format of cargo packets used in the authentication protocol, processed by CCM. *A* consists of the Janus header, the first byte of the ADB, and the flags. *N* is partitioned into both the baseline packet and the cargo, while *C* consists of all other values, except the CRC.

per packet may not pose a large obstacle, but it may still be considered depending on the nature of the mission.

3.2 Authenticated Encryption with AEGIS

A potentially even more suitable algorithm for use with Janus than CCM is AEGIS, proposed by Wu and Preneel [WP13]. Unlike CCM, which is a special mode of operation for a block cipher, AEGIS is a dedicated authenticated encryption algorithm, which also employs an underlying block cipher, but uses a message to update the state of the cipher. As with CCM, AES is most commonly used as the underlying block cipher and we also describe this version for our purposes.

There are three versions of AEGIS. AEGIS-128 uses a 128-bit key and 128-bit IV and processes 16-byte message blocks with 5 AES round functions. AEGIS-128L also uses a 128-bit key and 128-bit IV, but operates on 32-byte message blocks with 8 AES round functions. AEGIS-256 utilizes a 256-bit key and 256-bit IV, operates on 16-byte message blocks and uses 6 AES functions. For consistency with other relevant solutions, such as Venilia, we focus on the 256-bit key version, namely AEGIS-256. Note that, for the application in Janus, a shorter key could also be used, such as the one in AEGIS-128. This would slightly improve the performance of the algorithm.

AEGIS-256 defines a state update function, which updates an internal state using six rounds of AES. The main operations are the initialization, processing of the associated data, encryption, and finalization. Each of the operations updates the state and uses it to perform their respective tasks. The values of the variables used in the operations, when applied with Janus, are the following:

- $adlen$: the length of the associated data in bits (24 without cargo and 32 with cargo).
- $msglen$: the length of the plaintext and ciphertext in bits (34 bits per packet).
- $const$: a 32-byte constant set to: $00||01||01||02||03||05||08||0d||15||22||37||59||90||e9||79||62||db||3d||18||55||6d||c2||2f||f1||20||11||31||42||73||b5||28||dd$.
- IV_{256}: the 256-bit IV (obtained by repeating a 32-bit IV 8 times, similar to the CCM case).
- K_{256}: the 256-bit key.
- m_i: a 16-byte data block.
- S_i: the state at iteration i.
- t: the length of the authentication tag in bits (in our case 32).
- u: $u = \lceil \frac{adlen}{128} \rceil$.
- v: $v = \lceil \frac{msglen}{128} \rceil$.

Below, we give the details of the AEGIS algorithm as described in [WP13], with the use of the variable values specified above.

3.2.1 State Update

The state S_i in AEGIS-256 is a 96-byte variable which is updated for every 16-byte message block m_i that is processed. S_i consists of 6 16-byte elements $S_{i,j}$. In the state update function $S_{i+1} = \mathit{StateUpdate}(S_i, m_i)$, each element $S_{i,j}$ is updated independently until the new state S_{i+1} is derived. The function is defined as follows:

$$
\begin{aligned}
S_{i+1,0} &= \mathit{AESRound}(S_{i,5}, S_{i,0} \oplus m_i)\\
S_{i+1,1} &= \mathit{AESRound}(S_{i,0}, S_{i,1})\\
S_{i+1,2} &= \mathit{AESRound}(S_{i,1}, S_{i,2})\\
S_{i+1,3} &= \mathit{AESRound}(S_{i,2}, S_{i,3})\\
S_{i+1,4} &= \mathit{AESRound}(S_{i,3}, S_{i,4})\\
S_{i+1,5} &= \mathit{AESRound}(S_{i,4}, S_{i,5})
\end{aligned}
$$

Here, $\mathit{AESRound}(A, B)$ represents one round of AES, where A is a 16-byte state element and B is a 16-byte round key. The output is also a 16-byte state element.

3.2.2 Initialization

The initialization of AEGIS-256 consists of loading the 256-bit key and IV into the state, before updating the state 16 times using the key and IV as message. The whole initialization procedure is as follows:

1. The key and IV are loaded into the initial state S_{-16} combined with a predefined 32-byte constant, *const*:

$$
\begin{aligned}
S_{-16,0} &= K_{256,0} \oplus IV_{256,0}\\
S_{-16,1} &= K_{256,1} \oplus IV_{256,1}\\
S_{-16,2} &= \mathit{const}_1\\
S_{-16,3} &= \mathit{const}_0\\
S_{-16,4} &= K_{256,0} \oplus \mathit{const}_0\\
S_{-16,5} &= K_{256,1} \oplus \mathit{const}_1
\end{aligned}
$$

   Here, $K_{256,0}$ and $K_{256,1}$ are the first and second half of K_{256}, respectively. The same applies for *const*, where const_0 is its first half and const_1 is its second half.

2. The message m to be used in the initialization is constructed as follows:
   For $i = -4$ to -1:

$$
\begin{aligned}
m_{4i} &= K_{256,0}\\
m_{4i+1} &= K_{256,1}\\
m_{4i+2} &= K_{256,0} \oplus IV_{256,0}\\
m_{4i+3} &= K_{256,1} \oplus IV_{256,1}
\end{aligned}
$$

3. Finally, the state is updated 16 times, after which the initialization process is completed.
   For $i = -16$ to -1:

$$S_{i+1} = StateUpdate(S_i, m_i)$$

3.2.3 Processing of the Associated Data

After initialization, the associated data AD is used to update the state. AD is partitioned into 128-bit blocks, where the last block is padded with 0-bits if needed, until it is 128 bits long. The *StateUpdate* function is called for every block as follows:
For $i = 0$ to $\lceil \frac{adlen}{128} \rceil - 1$:

$$S_{i+1} = StateUpdate(S_i, AD_i)$$

3.2.4 Encryption

The encryption takes place for each plaintext block, where 16-byte plaintext blocks P_i are encrypted to ciphertext blocks C_i. As with the associated data, if the last plaintext block is shorter than 128 bits, it is padded with 0-bits to a length of 128. However, only the original plaintext, i. e., before padding, is encrypted. Thus, the length of the ciphertext to be transmitted across the channel, is not affected. The encryption occurs as follows:
For $i = 0$ to $v - 1$:

$$C_i = P_i \oplus S_{u+i,1} \oplus S_{u+i,4} \oplus S_{u+i,5} \oplus (S_{u+i,2} \& S_{u+i,3}),$$

where $u = \lceil \frac{adlen}{128} \rceil$ and $v = \lceil \frac{msglen}{128} \rceil$.

After producing one ciphertext block, the corresponding plaintext block is used to update the state:

$$S_{u+i+1} = StateUpdate(S_{u+i}, P_i).$$

3.2.5 Finalization

In the finalization stage, the authentication tag is constructed. The lengths of the associated data and the message, *adlen* and *msglen* (both 64-bit integers), as well as the values of u and v defined above, are used to further update the state. First, a temporary value is derived as follows:

$$tmp = S_{u+v,3} \oplus (adlen || msglen).$$

tmp is then used seven times to update the state:
For $i = u + v$ to $u + v + 6$:

$$S_{i+1} = StateUpdate(S_i, tmp).$$

The last state will then be S_{u+v+7}. The 16-byte elements of this state are XORed with each other to generate a stream of bits for the authentication tag:

$$T' = \bigoplus_{i=0}^{5} S_{u+v+7,i}.$$

The tag itself, T is chosen to be a desired amount of bits of T', starting from the MSB. In the Janus application, we use the first 32 bits of T.

3.2.6 Argumentation for the Use of AEGIS with Janus

Below, we give argumentation for why we chose to include AEGIS as a way of providing AEAD with Janus, in addition to CCM.

Historical Context and Alternative Algorithms

AEGIS was one of the winning submissions to the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) for authenticated encryption algorithms [Caesar]. CAESAR ran until 2019 and it had the purpose to replace current AE schemes, most notably CCM and GCM. Hence, we believe that an underwater solution based on AEGIS will keep underwater security schemes up to date with their counterparts above water, even if CCM and GCM get discontinued in WiFi networks and replaced by CAESAR candidates. In addition, AEGIS has already been deployed in surface-based autonomous vehicles that apply the Robot Operating System (ROS) [VSPF21]. Since the underwater environment has similar devices and characteristics as ROS-based networks, we believe the application of AEGIS under water is a natural next step.

There are three categories into which the CAESAR portfolio is organized, namely, lightweight applications for resource-constrained environments, high performance applications, and defense in depth. AEGIS was the winning candidate in the high performance category due to its efficient use of the AES round function, which has built-in hardware support by the Intel AES-NI instruction set. As a result, with such hardware support, AEGIS is much faster than AES in both CTR and CBC mode, which are both used in CCM.

As mentioned, AEGIS was not the winning candidate for resource-constrained environments, which is one of the main characteristics of our underwater application. Instead, the winning candidate in this category was ASCON [DEMS21]. However, ASCON is suited for resource-constrained environments in the context of small devices with low processing power. Its main focus is not the reduction of bandwidth consumption, which is our main goal. Although the reduction of computing resources is desirable underwater (especially in sensors), many underwater devices, such as AUVs are larger and usually have substantial processing power and memory. On such devices, the efficiency of the AES round function usage of AEGIS can be utilized, especially if hardware support for AES is provided. This in turn contributes to the reduction of the total time used by the authentication protocol, as the generation-encryption and decryption-verification operations are executed quickly.

Security Properties

Wu and Preneel [WP13] provide a security analysis of AEGIS in addition to its specification. They name three requirements for the secure operation of AEGIS:

1. Each key K used for initialization must be generated uniformly at random.
2. An IV must not be used more than once during the lifetime of a key and each key and IV pair must only be used with one size of the authentication tag T.
3. If the verification of the tag T fails, the decrypted plaintext and wrong T must not be disclosed to the public.

If these three requirements are satisfied, it is claimed that the success probability for a forgery of the MAC tag is $n \cdot 2^{-t}$, where n is the number of forgery attempts. If a forgery attack is not successful, the state and key cannot be recovered faster than exhaustive key search. For this reason, the recommended MAC tag length is at least 128 bits. However, to save bandwidth, we employ a 32-bit tag, as in the CCM-based protocols. Since 2^{32} forgery attempts are still infeasible to perform with the bit rate limitations of Janus, we believe that this length provides adequate protection against them.

3.2.7 Application in Janus-Based Communication

The protocol for authentication with AEGIS is very similar to the CCM version, described in Section 3.1.5, both with and without Janus cargo packets. The same values are transmitted in both schemes, resulting in an equal amount of bits. The nonce N from CCM is denoted IV here, but these values serve a very similar purpose. As with CCM, only a 32-bit IV is transmitted over the channel, and it is repeated 7 times at the receiving device to a length of 256 bits. The protocol without cargo can be seen in Figure 3.6, while the version with cargo is shown in Figure 3.7.

As with CCM, the use of cargo packets decreases the RTT of the protocol, but reserves the channel for the transmission of the cargo. As discussed in Section 3.1.5, this can be an obstacle in some network types.

3.3 Summary of the AEAD Schemes

In this chapter, two AEAD schemes for use in an authentication protocol based on Janus, were presented. Both provide strong confidentiality and integrity of data, as well as entity authentication, and the ranging functionality by Téglásy et al. Protocol versions with and without cargo were showcased, and both schemes have very similar communication protocols, resulting in practically identical bandwidth consumption.

Both CCM and AEGIS require a certain amount of processing and memory. For example, with CCM, it is necessary to perform operations on three 16-byte blocks for authentication and two 16-byte blocks for encryption. In the AEGIS solution, all operations involve the state variable, which is 96 bytes long. Due to the efficient usage of the AES round function of AEGIS, AEGIS has the potential to be the faster algorithm of the two, resulting in decreased RTTs of the authentication. However, since we are using AEGIS-256 and not AEGIS-128, a certain processing delay may still be added to the total RTT. Such time characteristics are discussed further in Section 6.2.

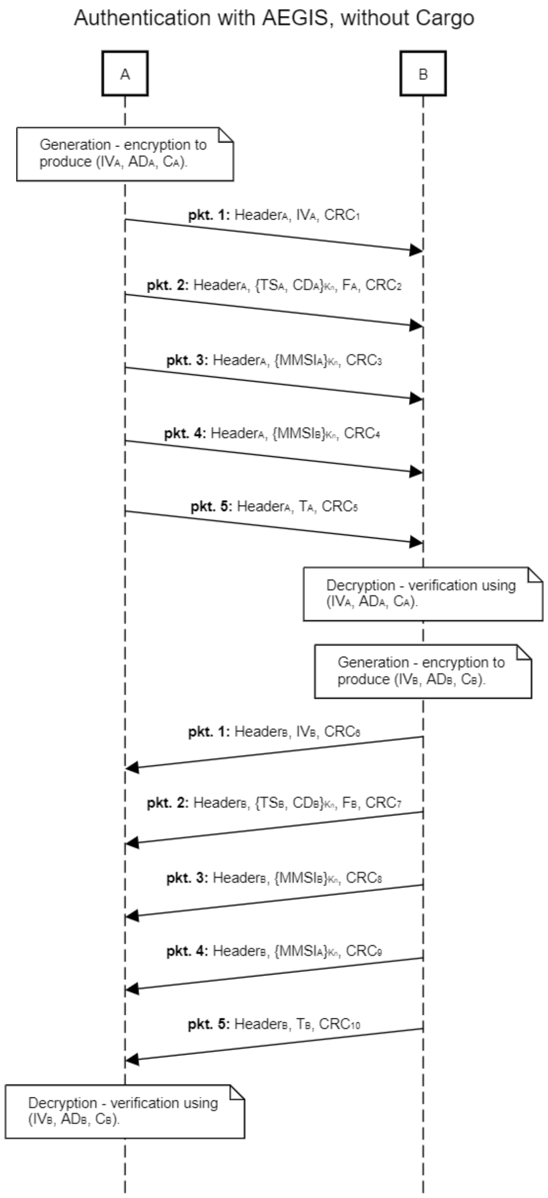

Authentication with AEGIS, without Cargo
A
B
Generation - encryption to produce (IV_A, AD_A, C_A).
pkt. 1: $Header_A$, IV_A, CRC_1
pkt. 2: $Header_A$, $\{TS_A, CD_A\}_{K_n}$, F_A, CRC_2
pkt. 3: $Header_A$, $\{MMSI_A\}_{K_n}$, CRC_3
pkt. 4: $Header_A$, $\{MMSI_B\}_{K_n}$, CRC_4
pkt. 5: $Header_A$, T_A, CRC_5
Decryption - verification using (IV_A, AD_A, C_A).
Generation - encryption to produce (IV_B, AD_B, C_B).
pkt. 1: $Header_B$, IV_B, CRC_6
pkt. 2: $Header_B$, $\{TS_B, CD_B\}_{K_n}$, F_B, CRC_7
pkt. 3: $Header_B$, $\{MMSI_B\}_{K_n}$, CRC_8
pkt. 4: $Header_B$, $\{MMSI_A\}_{K_n}$, CRC_9
pkt. 5: $Header_B$, T_B, CRC_{10}
Decryption - verification using (IV_B, AD_B, C_B).

Figure 3.6: AEGIS used in the authentication protocol, without the usage of cargo packets.

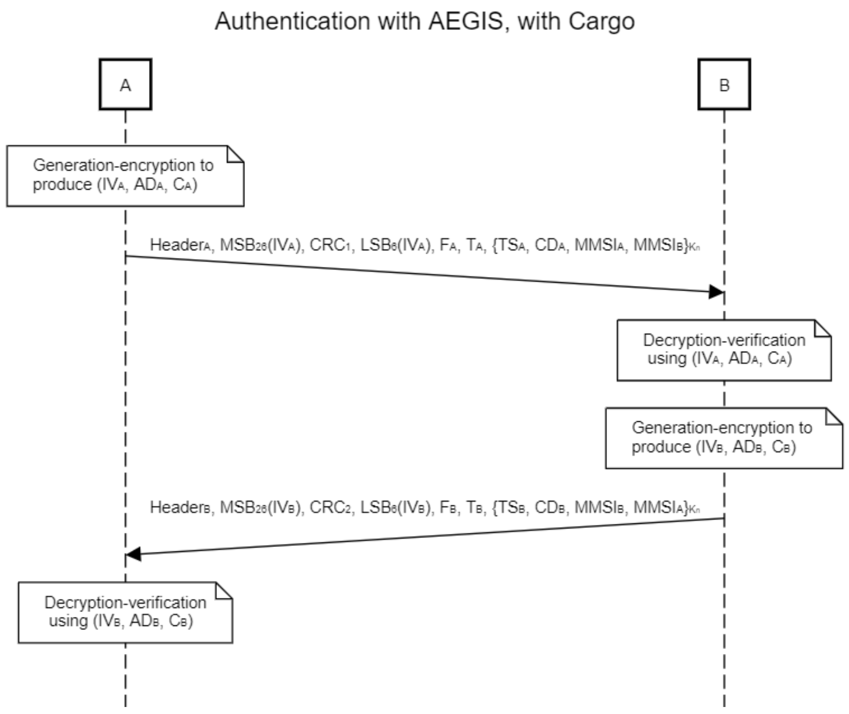

Authentication with AEGIS, with Cargo
A
B
Generation-encryption to produce (IV_A, AD_A, C_A)
$Header_A$, $MSB_{26}(IV_A)$, CRC_1, $LSB_6(IV_A)$, F_A, T_A, $\{TS_A, CD_A, MMSI_A, MMSI_B\}_{K_n}$
Decryption-verification using (IV_A, AD_A, C_A)
Generation-encryption to produce (IV_B, AD_B, C_B)
$Header_B$, $MSB_{26}(IV_B)$, CRC_2, $LSB_6(IV_B)$, F_B, T_B, $\{TS_B, CD_B, MMSI_B, MMSI_A\}_{K_n}$
Decryption-verification using (IV_B, AD_B, C_B)

Figure 3.7: AEGIS used in the authentication protocol, with the usage of cargo packets.

Chapter 4 Security Analysis of the Proposed AE Solutions

In this chapter, we assess the level of security provided by the AE methods proposed in Chapter 3. For the assessment, the STRIDE threat modelling scheme [SCO+18] is used. A model generated with this approach provides an overview of the possible ways that attackers can disrupt the communication among devices and cause harm, as well as the level of protection that the security schemes provide against these threats. The general threat landscape is assumed to be as described in Section 2.4. STRIDE consists of determining the adversaries' capabilities of performing spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege. Following are descriptions of how each of these goals would be achieved in the underwater environment and the consequences they would have. Additionally, we show how our solutions described in Chapter 3 mitigate these attacks.

4.1 Spoofing

Spoofing refers to impersonating something or someone else and it is a sign that authentication is not provided. Normally, attackers would spoof the identity of a legitimate node to successfully authenticate themselves to other legitimate nodes and establish secure communication with them. This allows them to mount and execute larger attacks.

In our authentication solutions, we claim that it is infeasible to spoof the identities of legitimate nodes. In our threat model, we follow Kerckhoff's principle and assume that adversaries know all details about our cryptosystem, except the decryption key. Based on this, it is difficult for adversaries to spoof the identities of any legitimate nodes because all identities in the protocol are transmitted in encrypted form. Therefore, if an attacker aims to transmit a spoofed identity in a legitimate message, the identity must be encrypted. If a legitimate node receives an identity in cleartext, the message will be ignored.

There are, however, attacks that adversaries can launch that do not involve spoofing of identities directly, but rather other data elements. One such attack is ACK spoofing, described below.

4.1.1 ACK Spoofing

ACK spoofing is described in [Dom11] and mentioned in Section 2.4. In this attack, an attacker sends illegitimate ACK messages through a low-quality channel that is prone to packet and connectivity loss. This way, legitimate nodes that receive the ACK messages will deem this channel to be reliable, when it is not. The consequence is that they will continue to send data through this channel, resulting in increased data loss and disruption of the communication.

A way to perform ACK spoofing in our solutions in Chapter 3 is to spoof F. F consists of a SYN flag and an ACK flag, which indicate the status of the authentication protocol. In the first stage of the protocol, when A sends its information to B, SYN = 1 and ACK = 0. When B responds, it is still the case that SYN = 1, but also, ACK = 1, to indicate that the previous message or messages were successfully received. When A receives this acknowledgement, it will deem the channel reliable and secure and will further use it to send messages to B after the authentication. Using this fact, an attacker D can detect when A initiates the protocol by overhearing it, then send spoofed ACK flags back to A, even if the channel is unreliable and A's messages were in fact lost on their way to B. A will then continue to send messages through this channel, leading to more data loss. A conceptual view of this attack is shown in Figure 4.1.

Although an attack like this is theoretically possible and relatively common in UWCNs, it is not likely that it will affect our solutions. This is because D needs to include many other data elements in addition to F in order to generate a legitimate message in the protocol. A message with only the plaintext representation of F is not a legitimate message in any protocol version and will therefore be ignored. For instance, in the protocol versions without cargo, F is sent together with TS and CD in encrypted form in packet 2. This means that D in addition needs to forge valid TS and CD values and encrypt them with the secret key. In the versions with cargo, D would need to obtain all data elements involved in the protocol for a message to be accepted as authentic. Consequently, it is infeasible for an adversary to perform this attack and our solutions provide strong protection against it.

4.2 Tampering

Tampering refers to modifying data or code in any environment. It affects the integrity property. Generally, our solutions are more vulnerable to tampering than spoofing, as attackers can intercept and modify data in the Janus packets as they

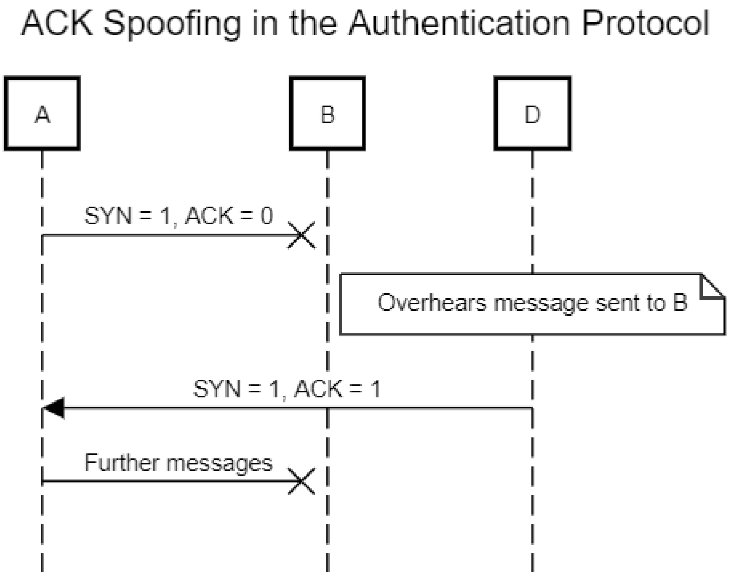

ACK Spoofing in the Authentication Protocol
A
B
D
SYN = 1, ACK = 0
Overhears message sent to B
SYN = 1, ACK = 1
Further messages

Figure 4.1: An attacker D sends a spoofed ACK flag to A, causing A to continue sending messages through an unreliable channel.

traverse the channel. Still, our solutions provide relatively strong protection against tampering of intercepted messages on the channel due to the requirement of physical access to a device and/or a faulty Random Bit Generator (RBG). The reasons for this are explained later in this section. Tampering is also usually the basis for mounting other kinds of attacks. For example, ACK spoofing, described in the previous section, can also be performed by intercepting packets during propagation time and modifying the F field. Following are descriptions of other possible tampering attacks.

4.2.1 Nonce and IV Modification

Since the nonce in CCM and IV in AEGIS are used in their respective generation-encryption and decryption-verification processes, the tampering of them will cause errors. These values are also sent in the clear in both algorithms, meaning that adversaries that intercept them can easily change them to predictable values. They can also inject previously recorded nonces and IVs, thereby forcing their reuse. Since it is generally very important that such counter values are unique during the lifetime of a key, this can potentially pose a serious threat.

As described in Section 3.1, in CCM, the nonce N is used during both the generation-encryption and decryption-verification operations. In both cases, it is used to generate the initial block B_0 in the CBC-MAC algorithm (see Table 3.1), as well as each counter block Ctr_i during the CTR mode encryption (see Table 3.3). In

AEGIS, the IV is also used in both generation-encryption and decryption-verification. It is used to initialize the state, which is later used for further processing.

Generation-Encryption

We first consider generation-encryption. In CBC-MAC, N is the only source of unpredictability. In our protocol, the tag length t, the nonce length n, and the length of Q are always the same, meaning that the uniqueness of B_0 depends entirely of N. Thus, if N is replayed, the effects will propagate until the last keystream block Y_r due to the CBC mode of operation. If N is repeated enough times, a certain predictability of the value of T will be created, making it possible to forge a valid T with less complexity than the birthday paradox, outlined in Section 3.1.4. In CTR encryption, each Ctr_i consists of a set of flags, N, and an additional counter i for every block. Both the flags and N are constant in all Ctr_i, while i is incremented for each block. This means that, again, N makes each encryption procedure unique, as the value of i is never random. Consequently, if N is repeated, the resulting counter blocks and keystream will be predictable.

The case of AEGIS is similar. During initialization, the IV is the only unique variable, as the key and the $const_0$ and $const_1$ variables are all constant. The state after initialization is used to process the associated data, encrypt the plaintext, and produce the MAC tag. Hence, a predictability caused by the reuse of an IV will propagate through the entire process and potentially allow an attacker to recover the state with statistical attacks.

A general overview of the effect of nonce or IV reuse is shown in Figure 4.2.

Attacks of this kind would greatly increase an adversary's chance of breaking the cryptosystem. However, this implies that the adversary manages to inject earlier nonce values such that they are used in practice. In our CCM-based protocols, since all values that are used as input to the generation-encryption process, namely, N, AD, and P, are generated locally on a device, there is no possibility that nonce reuse can be forced by tampering with the communication channel. Similarly, the input to AEGIS are the IV, AD, and P, which are generated locally. Therefore, reuse of N or the IV can only be achieved if an adversary manages to physically capture a device and inject N manually. Alternatively, the device can employ a faulty RBG, such that collisions occur. We deem the probabilities of both these events small. Thus, cryptanalysis based on nonce or IV reuse on the sending device is not a serious threat.

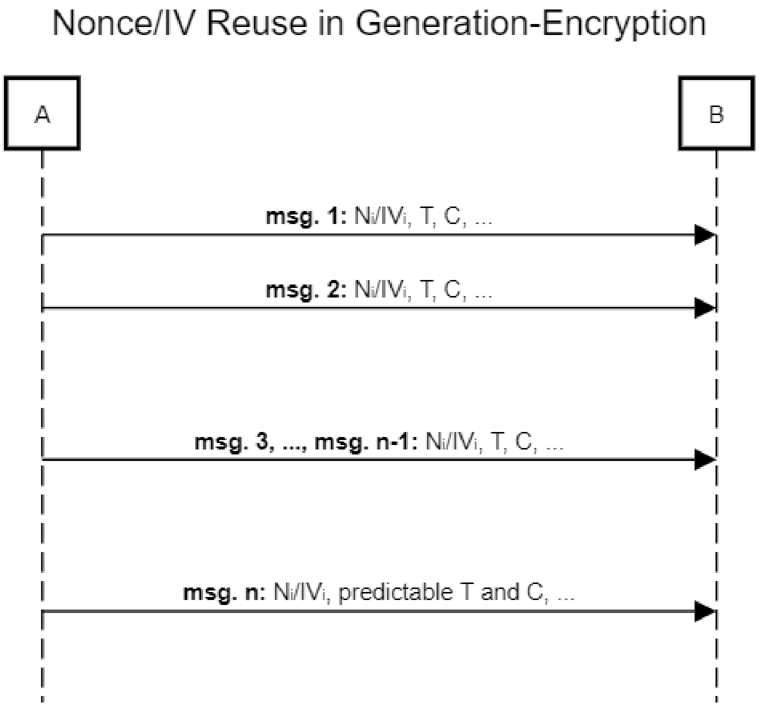

Nonce/IV Reuse in Generation-Encryption
A
B
msg. 1: Ni/IVi, T, C, ...
msg. 2: Ni/IVi, T, C, ...
msg. 3, ..., msg. n-1: Ni/IVi, T, C, ...
msg. n: Ni/IVi, predictable T and C, ...

Figure 4.2: After enough messages sent with the same nonce or IV under one key, the authentication tag and ciphertext become predictable.

Decryption-Verification

Next, we consider decryption-verification. It is more likely that N or the IV will be replayed in decryption-verification because it is more feasible for an attacker to inject these values into intercepted packets and forward the modified packets to the designated receiver. The attack would be performed by combining eavesdropping and jamming, as described in Section 2.4.1, or using a wormhole link, described in Section 2.4.2. This way, the attacker could ensure that the intercepted message is modified and that the legitimate message does not reach the receiving device. If no cargo is used, the attacker would intercept and modify packet 1. If cargo packets are used, an entire packet would be intercepted and the nonce or IV would be replaced in it. Since the nonce and IV are transmitted in cleartext, the attacker does not need the decryption key to successfully replace them in the intercepted data with any value. The concept of this attack is shown in Figure 4.3.

In CCM, since N is used to generate the counter blocks Ctr_i and the first authentication block B_0 in the decryption-verification operation, the modification of N will lead to the generation of different Ctr_i and B_i than what was generated by the

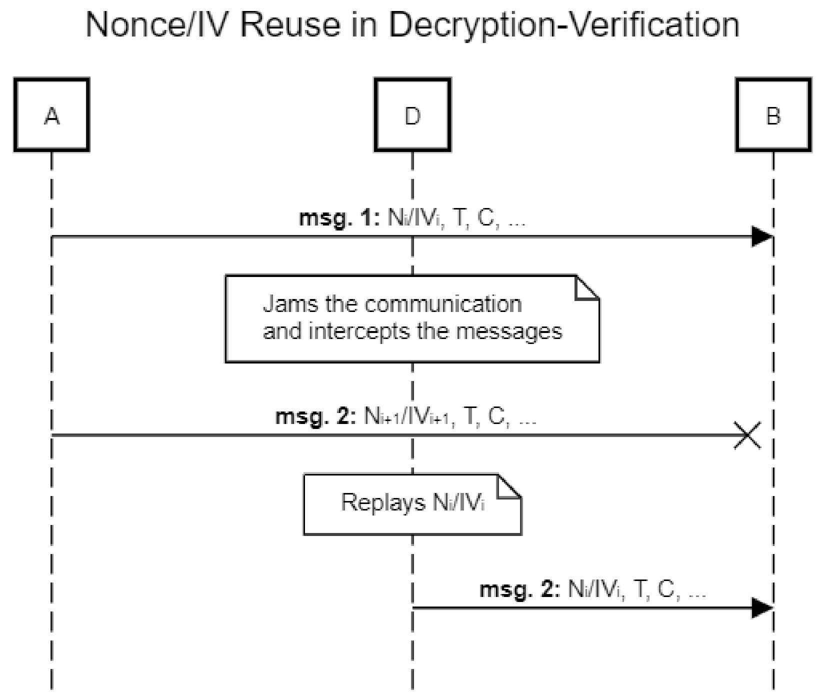

Nonce/IV Reuse in Decryption-Verification
A
D
B
msg. 1: Ni/IVi, T, C, ...
Jams the communication and intercepts the messages
msg. 2: Ni+1/IVi+1, T, C, ...
Replays Ni/IVi
msg. 2: Ni/IVi, T, C, ...

Figure 4.3: A malicious node D intercepts the first nonce or IV i, then jams the communication, such that the next message does not reach B, while also intercepting the next message. It then replays N_i or IV_i.

sending device. This will lead to the derivation of a different plaintext than the one that was encrypted by the sending device, meaning that both the decrypted payload and MAC tag will be different. It will also lead to the recomputation of a different MAC tag with the CBC-MAC algorithm. The tag verification in CCM consists of checking whether the decrypted tag from the received ciphertext equals the tag that is recomputed using CBC-MAC. In our case, if a valid nonce N_1 is replaced in transit with another nonce N_2, the decrypted tag that is derived from the ciphertext will not equal the tag that is recomputed, even though the same invalid nonce N_2 is used in the decryption and the CBC-MAC recomputation. This is because the triplet (N, AD, P) is used as input to the formatting function that produces the blocks B_i. During decryption, the obtained payload P is different than the payload sent by the originating device because a different nonce was used to generate the counter blocks Ctr_i. Because of this, the wrong P is used to recompute T at the receiving end, resulting in a mismatch between the decrypted T and the recomputed one. Hence, the verification process will return the value $INVALID$, and the authentication will fail.

AEGIS provides a similar functionality. The IV is used to initialize the state, from which, ultimately, the MAC tag is derived. If the IV is changed in transit, the reconstructed tag will be different from the one that was received and the authentication will fail. Additionally, the decryption of the ciphertext will not result in the original plaintext, as the state after initialization is used for decryption.

Consequently, the modification of N or the IV in transit in an MitM attack is no major threat to our protocols. The design of the decryption-verification operations in CCM and AEGIS take this attack into account and the attack triggers an authentication failure. When the authentication fails, the received information is dropped and no security properties are removed. Still, such attacks can cause longer delays in the authentication procedures due to failed protocol runs and message re-transmissions.

4.2.2 Modification of the Associated Data and the Ciphertext

In CCM, the modification of AD or C in transit will have the same effect as the modification of N described above. Again, this is because both AD and P are input to the CBC-MAC algorithm. If the designated device receives a different AD from that which the originating device sent, the recomputed tag T will be different than the one received, resulting in an authentication failure. Similarly, if C is modified, its decryption will lead to a different P or T, or both. The tag that is then recomputed cannot match the received one, and the authentication will again fail.

Similarly, the MAC tag reconstruction in AEGIS takes place after the state updates done in the processing of AD and the decryption. Consequently, if AD or C are modified in transit, the reconstructed T will be different than the one received. If T itself is modified in transit, the authentication will also fail.

4.2.3 Modification of the CRC

In both the CCM and AEGIS-based solutions, the CRC is the only element that is not part of N, the IV, AD, or C. Hence, it is not protected in any way. Adversaries can therefore modify the CRC in transit and the modification will not get detected by the security algorithms.

If the CRC is modified, the receiving device will interpret the packet it receives as erroneous. It is up to the device to decide how many errors in the packet it is willing to accept before the packet is dropped. The CRC is calculated by dividing the polynomial generated from its preceding 56 bits in the baseline packet, with the polynomial $p(x) = x^8 + x^2 + x + 1$, as shown in Table 2.2. Hence, the modification of only a few bits may still be accepted by the receiving device. However, an adversary that has intercepted a packet has the possibility to complement all 8 bits in the CRC, causing the packet to be rejected. In other words, an adversary that modifies the CRC has the possibility to drop packets at will.

This kind of adversarial interference can lead to reduced availability, but it will not result in the compromise of sensitive data. This is because the packets are simply dropped by the designated receiver, and the adversary cannot extract the encrypted information from the packet. Since this attack is very similar to jamming, similar actions can be taken against it as they are taken against jamming attacks. According to [Dom11], devices under such attacks should save their power and occasionally transmit data to check if the attack is finished. In our case, the authentication queries happen intermittently every 5 to 10 minutes. Hence, if one query fails due to packet loss, another attempt will be made after a short time. The queries can be repeated until the attack is over and the authentication succeeds.

4.3 Repudiation

Performing repudiation means to claim not to have performed an action or not to be responsible for an event. As non-repudiation is a common security property that must be satisfied by most systems, it is necessary to provide measures that prevent entities from denying certain actions, or that minimize the harm caused by such denial. According to the security services defined in [TC91], there are two types of non-repudiation, namely, non-repudiation of origin and of destination. They define that entities should not be able to deny sending or receiving a message, respectively. In the authentication methods proposed in this thesis, such denial would be done by the inclusion of incorrect MMSIs in messages. However, as described in Section 4.1, this is infeasible to perform for adversaries. Nevertheless, there are theoretical ways in which denial of participation in the authentication protocol might be performed, as described below.

4.3.1 Denial of Protocol Participation through Flag Manipulation

By modifying F and setting the SYN flag to 0, a malicious device can claim that it has not initiated the authentication protocol, even if the rest of the messages it sends have the correct format according to the protocol specifications. This will cause the device that receives these messages to not interpret them as part of the protocol and, thus, not decrypt and verify the received data or issue a response. Similarly, a malicious receiving device may claim not to have received any messages by transmitting a response with the ACK flag set to 0 or not responding at all. In other words, this kind of misuse of flags can create messages that are undefined in the authentication protocols, thereby causing the authentication procedure to fail, or otherwise disrupting the communication network.

This attack is very similar to ACK spoofing, described in Section 4.1.1 above. Hence, it is equally infeasible to perform, as an attacker willing to send illegitimate values of F would need to obtain all other data elements in the protocol, as well as K_n, in order to create valid messages. The attacker cannot intercept and modify F in a message in transit either since F is included in the associated data AD. As described in Section 4.2.2, this will result in an authentication failure. Consequently, repudiation through flag manipulation is not a serious threat to our authentication solutions.

4.4 Information Disclosure

Information disclosure involves exposing information to someone not authorized to see it, and it is an attack against confidentiality. Generally, in large systems, information disclosure occurs when confidential data in databases or on network connections gets accessed by attackers and released to the public or otherwise misused. In our case, the only data to access are the lookup tables of MMSIs on devices, which are not regarded confidential, and the secret key. It is therefore important that the key is kept secret. On the other hand, information sent in packets among devices may still be disclosed if an attacker manages to bypass the encryption. In the case of CCM, this would involve obtaining the keystream blocks S_i by brute force or other forms of cryptanalysis of AES. With AEGIS, it would involve recovering the secret key or the internal state variable by the same means.

4.4.1 Disclosure of Information on the Channel

As discussed in Section 3.1.4, if adversaries are to bypass the confidentiality of CCM, they must execute birthday attacks or find an internal collision in the CBC-MAC computation. Similarly, as mentioned in Section 3.2.6, the state and key of AEGIS cannot be recovered faster than exhaustive key search. As described in Section 4.2.1, these properties rely on secure generation of nonces and IVs. If nonce and key pairs are not reused in CCM, and IV and key pairs are not reused in AEGIS, there is little risk of disclosure of confidential information from data sent across the communication channel.

4.4.2 Disclosure of Locally Stored Information

A considerable threat of information disclosure arises if adversaries obtain an entire device, such as an AUV, and access its onboard storage. Since the secret key is among the data elements stored on the devices, this would give adversaries the possibility to decrypt all intercepted data encrypted with that key, as well as to forge new legitimate packets. A mitigation in this situation is more frequent renewal of the pre-shared long-term key.

4.5 Denial of Service (DoS)

DoS would eliminate the availability property, and it involves denying or degrading service to users. According to [TWPK], a DoS may be caused through repeated retransmission of messages, as well as modification of ciphertexts and CRCs. In our case, adversaries can also modify nonces, IVs, and the associated data, to provoke failed authentications. These attacks are described in earlier sections of this chapter. An attacker may also incapacitate a legitimate acoustic signal by jamming, as described in Section 2.4.1.

As mentioned in Section 4.2.3, devices under DoS attacks should save their power and wait until the attack is over. Generally, if an authentication attempt fails, another will be made after maximally 10 minutes. Hence, there will be more opportunities to perform the authentication successfully. Until this point, the devices that attempt the authentication will not communicate with each other, as the packets exchanged between them will be lost. Nevertheless, this will not lead to the compromise of confidential data, making the consequences of the DoS sustainable.

4.6 Elevation of Privilege

Elevation of privilege is related to authorization and it involves not permitting someone to do something they are not authorized for. In the context of the authentication protocols, legitimate devices should not perform other computations or transmit other messages than what has been defined. Additionally, adversaries should not be allowed to obtain the key, as this would provide them the privileges they need to masquerade as legitimate devices and disrupt the network. If the key is compromised, a key renewal should be considered, as mentioned above.

4.7 Summary of the Security Analysis

After performing the STRIDE-based analysis, we conclude that the authentication methods proposed in this thesis provide a high level of protection against the majority of attacks in the STRIDE categories. The only attacks that have a high chance of breaking important security properties are those against which no protocol or algorithm can provide protection, such as DoS or physical capture of devices.

Chapter 5

Methodology

This thesis fulfills design goals, but it also answers knowledge questions. The knowledge questions are related to acoustic communications and the assessment and modelling of various AE schemes for use with Janus. The design goals have been to investigate which security properties are needed in the relevant communication scenarios, determine which solutions provide these properties, and design an appropriate AE scheme based on the findings. As such, the objectives and research questions of this thesis are related to both traditional science and design science.

5.1 Design Cycle and Empirical Cycle

Since it was not known in advance which AE scheme is appropriate and satisfies all requirements, the research process was based on identifying schemes that show promise, selecting candidates, and possibly adapting them to the Janus-based underwater context. The process of designing an AE scheme and, thus, fulfilling the design goals, took the form of a design or engineering cycle, as described by Wieringa [Wie14], which divides the methodology into problem investigation, treatment design, treatment validation, treatment implementation, and implementation evaluation. A treatment in this context refers to the proposed AE scheme. Of these parts, we have used only the first three, as the treatment implementation would assume that we implement our designs in hardware. This is beyond the scope of this thesis. The design cycle is shown in Figure 5.1.

In this thesis, the different stages of the design cycle consisted of the following:

1. **Problem investigation:** The investigation of the problem consisted of a cryptographic analysis proposed by Téglásy et al., which lead to the discovery of its limitations (Section 2.8.1). In addition, a review of the security requirements of underwater communication was performed (Section 2.5), which identified

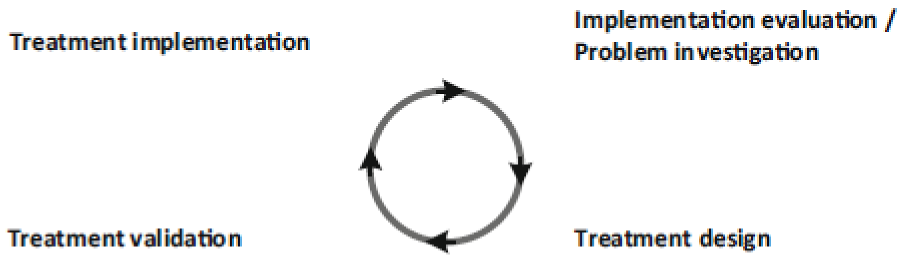


Figure 5.1: The stages in the design cycle. Adapted from [Wie14].

authenticated encryption as the main mechanism that was required of the security schemes that were to be proposed.

2. **Treatment design:** As the AEAD algorithms used in this thesis were already implemented in other environments, the treatment design consisted of adapting these algorithms to Janus and designing the protocols and packet formats that were used. This included defining all computations related to the protocols, such as the repetition of nonces and IVs at the receiving devices.

3. **Treatment validation:** The proposed schemes were validated by the security analysis (Chapter 4), which analyzes to which extent the schemes satisfy the security requirements. Additionally, it was determined that the new schemes still allow the performance of ranging, which was one of the main functions of the protocol by Téglásy et al.

The knowledge questions were answered in an empirical cycle, shown in Figure 5.2, also described by Wieringa. The stages in this cycle are the following:

1. **Research problem analysis:** The research problem analysis consisted of the threat landscape modelling and determination of the underwater security requirements (Section 2.4 and Section 2.5).

2. **Research and inference design:** The conclusions were obtained by determining the appropriate AE schemes, implementing them, and measuring the time needed for the authentication protocols to complete. Additionally, an analysis of identified vulnerabilities and attacks on the protocols was performed and the security level of the protocols was assessed (Chapter 4).

3. **Validation and research execution:** The experiments were performed as defined in the research and inference design stage.

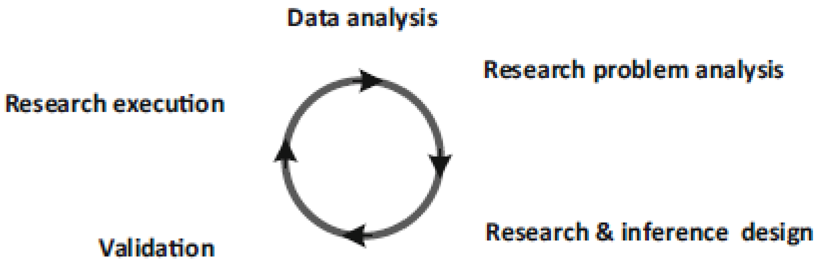


Figure 5.2: The stages in the empirical cycle. Adapted from [Wie14].

4. **Data analysis:** The results of the simulations and security analysis were assessed, presented, and discussed.

5.2 Summary of the Methodology

The methodology of the execution of this project involved both design science and traditional science, as both design and knowledge questions were answered. The design and investigation stages were performed by reviewing the threat landscape and communication requirements, such that adequate security schemes could be identified. Experiments were then conducted to answer the knowledge questions, which were related to the time spent for authentication and the security provided in the reviewed threat landscape. The experiments and results are discussed further in the next chapter. Overall, the different stages of the methodology are connected by a general need to secure the underwater authentication process. The analysis and experiments conducted ensure trustworthy results.

Chapter 6

Experimental Work, Results, and Discussion

In the experimental work in this thesis, the authentication protocols in Chapter 3 were simulated and their RTTs were measured in a simulation environment. The computation time of CCM and AEGIS that takes place locally on the sending and receiving devices was also measured in a simulator. This chapter describes how the simulations were conducted, presents the results obtained in these simulations and provides observations and comments on them.

6.1 Experimental Work

A simulation of underwater networks with authenticated encryption algorithms was implemented in C and C++ using the UAN library of the NS3 network simulation framework [NS3] and the existing implementation of Janus found on the Janus Wiki [JanusWiki]. The source code of the simulation scripts is available in Appendix B. The implementation and experiments were conducted in an Ubuntu-based 64-bit Linux Mint virtual machine with two processor cores and 4096 MB of base memory.

6.1.1 Experimental Setup

The UAN library was used to simulate nodes in an underwater network that send information to each other using acoustic waves. The nodes that were simulated represent AUVs that would communicate in a similar manner in the real world. The UAN library and the Janus simulation were originally implemented separately by different authors, and we combined them to create an underwater network simulation in which Janus packets are sent, thereby resembling communication with Janus as the underlying standard. The Janus implementation by itself only simulates transmission and reception at transmitting and receiving devices, respectively, without giving information about the transmission channel itself and with no possibility to run a simulation over time. A sound wave is modulated as per Janus specification, which transmits a packet. The packet is stored in a file, which the receiving device simulation reads and demodulates. To utilize the channel simulations and longer simulation periods that are provided by NS3, the packet data produced by the Janus simulation was stored into files, which were read into a simulation script in the UAN library. The data was then used conventionally in the UAN library, using its built-in packet objects to transmit the data across the simulated network.

Janus Packet Simulation

The Janus simulation allows for the creation of one packet at a time, where the packet is first populated with data and modulated, before the resulting sound wave that represents the packet is saved to a file. The file is then read and demodulated, and the data is extracted and printed. The packet writing and reading are performed by two different modules, called Janus Tx and Janus Rx, which represent a transmitting and a receiving device, respectively. When working with the C implementation, the Tx and Rx modules are called from the command line, for which examples are provided in the Janus README document [Zap]. An example command for creating a Janus packet with no application data is as follows:

```
janus-tx --pset-file janus-c-3.0.5/etc/parameter_sets.csv
--pset-id 1 --stream-driver-args /tmp/test_janus.wav --verbose 2
```

The output will contain information about the configuration options, the parameter set, the output stream, the state, and the packet fields themselves. The packet fields in the output alone for the command above are shown in Table 6.1. We observe that all the fields from the bit allocation table of the Janus specification, Table 2.2, are given, from the version number to the CRC. The binary representation of the whole packet was saved to a text file, such that it could be read in a script in the UAN simulation.

NS3 UAN Simulation

The UAN library of NS3 allows to simulate the sending of packets of arbitrary length and content among underwater nodes over arbitrary periods of time. It provides two propagation models for acoustic waves under water: an ideal channel model with a high range and the Thorp model which describes the waves' path loss over time, thereby reducing the range of the signals. Compared to other propagation models, such as the Monterrey Miami Parabolic Equation (MMPE) and Bellhop models, the Thorp model is of low complexity, but also of relatively low accuracy. This is because it only takes into consideration the general attenuation of sound waves as they propagate through water particles and ignores factors such as water temperature, salinity, wave activity, etc. [LM12]. On the documentation pages of NS3, it is stated that a simulation environment using the much more accurate Bellhop

Table 6.1: Output of the creation of an empty Janus packet

Bytes (decimal)	| 50| 16| 0| 0| 0| 0| 0| 85|
Bytes (hex)	| 32| 10| 00| 00| 00| 00| 00| 55|
Fields (binary)	|0011|0|0|1|0|00010000|000000| 0000000000000000000000000000000000| 01010101|
Version Number (4 bits)	3
Mobility Flag (1 bit)	0
Schedule Flag (1 bit)	0
Tx/Rx Flag (1 bit)	0
Forwarding Capability (1 bit)	0
Class User Identifier (8 bits)	16 (NATO JANUS reference Implementation)
Application Type (6 bits)	0
Application Data (34 bits)	00
Cargo Size	0
CRC (8 bits)	85
CRC Validity	1

model is currently being implemented, but it has not been published. Consequently, the Thorp approximation was used in our simulations.

Given the simplistic propagation model, the speed of sound is assumed to constantly be 1500 m/s. The theoretical delay in the transmission of messages can then be calculated as $delay = \frac{distance}{speed}$. Using a similar approximation, the series of events shown in Figure 6.1 were simulated, where the propagation time increases proportionally to the distance between the devices. However, as discussed in Section 6.3, the simulated transmission delay is not exactly the same as the theoretical one.

Two main experiment scenarios were set up: in one scenario, only the range limitations of the Thorp model were taken into account, while in the other, a noise of varying degree was added to the communication channel. In the case without noise, every attempted run of the protocols was successful and no packet loss occurred. When noise was added, packets were dropped in transmission, simulating DoS attacks and/or poor channel quality. As both CCM and AEGIS need all data correctly transmitted to the receiving device for decryption-verification to take place, the loss of a single packet would be enough for the protocols to fail. The probability of packet loss was represented by a random integer, which determined when the protocol would fail. On each protocol failure, 7 minutes were added to the total protocol RTT, as the simulator was configured to run the protocol every 7 minutes.

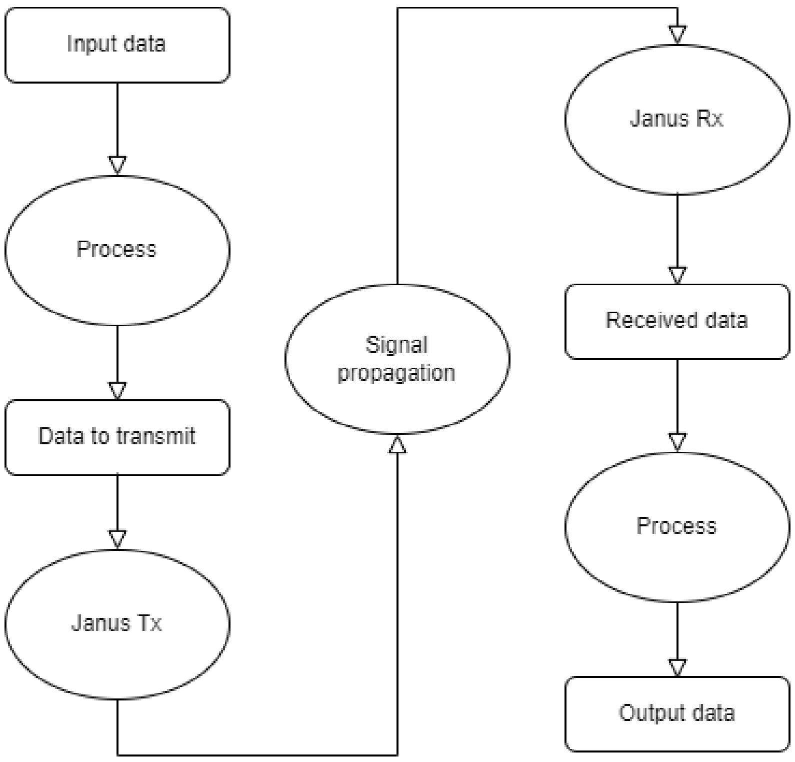


Figure 6.1: The pipeline of processes simulated in NS3. The circles represent time-consuming processes, while the rectangles represent intermediate values. The figure shows the process of sending data in one direction only.

Generally, the protocols would be run every 5 to 10 minutes in the real world, making this simulation a nearly average case. The packet loss probability was varied from 0.1 to 0.9, resulting in different numbers of retries of the protocol execution and, thus, different RTTs.

6.1.2 Implementation of the Security Algorithms

The RC5, CCM, and AEGIS algorithms were implemented in C, following their specifications in [Riv95], [Dwo04], and [WP13], respectively. For CCM and AEGIS, a ready-made lightweight implementation of AES was used. The correct implementations of the algorithms were verified using the test vectors provided in all three specifications.

The algorithms were then incorporated into the Janus simulation. For this, the original version of Janus was modified such that the functions of the security

Table 6.2: Output of the creation of a Janus packet containing $MMSI_A$ in encrypted form, in the CCM-based protocol.

Bytes (decimal)	| 48| 0| 3| 66|130| 63|108|135|
Bytes (hex)	| 30| 00| 03| 42| 82| 3F| 6C| 87|
Fields (binary)	|0011|0|0|0|0|00000000|000000| 1101000010100000100011111101101100| 10000111|
Version Number (4 bits)	3
Mobility Flag (1 bit)	0
Schedule Flag (1 bit)	0
Tx/Rx Flag (1 bit)	0
Forwarding Capability (1 bit)	0
Class User Identifier (8 bits)	16 (NATO JANUS reference Implementation)
Application Type (6 bits)	0
Application Data (34 bits)	00
CRC (8 bits)	135
CRC Validity	0

algorithms were called with their respective portions of the Janus packet as input. For example, in the case of CCM, the nonce, associated data, and plaintext of the entire protocols in Section 3.1.5 was generated and passed to the generation-encryption function. The resulting output was then partitioned into five baseline packets and one cargo packet for the versions without and with cargo, respectively. The partitioning was done by invoking the builtin functions of the Janus simulation for creating new packets, extracting and modifying the ADB, as well as other utilities. The simulation was prepared such that the same `janus-tx` command shown above created all packets needed to transmit for each protocol version. For example, the third packet in the CCM-based protocol without cargo, containing $MMSI_A$ (see Figure 3.3), was represented with the output given in Table 6.2. Note the `Bytes (decimal)`, `Bytes (hex)`, `Fields (binary)`, and `CRC (8 bits)` fields, showing the data in the processed packet.

6.1.3 Work with Acoustic Modems

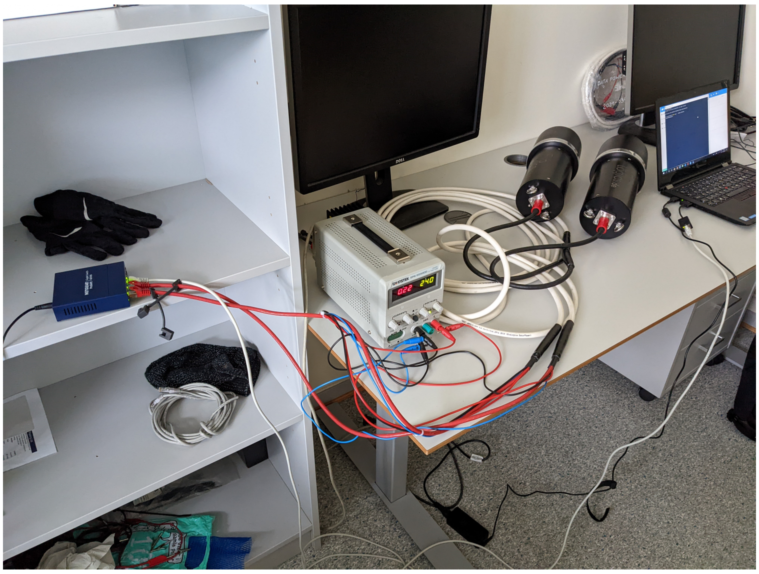
To determine the level to which our proposed protocols can be implemented and used in reality, work was conducted towards an implementation on two EvoLogics S2CR acoustic modems. These modems are compatible with the Janus standard and allow for the transmission of information with it. In our setup, they were configured for in-air testing and were used in a laboratory environment, outside water. A cable connection over Ethernet was used to connect the modems to a computer, from which commands were sent to configure the Janus Tx and Rx servers. The setup is shown in Figure 6.2.

The commands to the modems were issued through the Software Defined Modem (SDM) shell, which allows for transmission of data from a computer to the modems via a Transmission Control Protocol (TCP) connection. Through this connection, Janus commands, such as the one in Section 6.1.1, were used to transmit a packet, as well as to initiate listening processes on the modems on their respective TCP ports. The sound file that was generated by Janus Tx was then played from one of the modems.

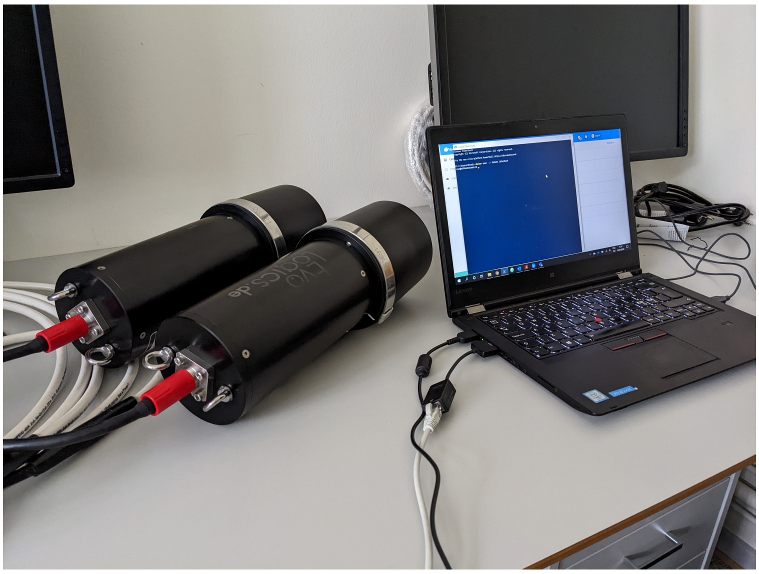
A full implementation of the protocols was not performed, as such an implementation would require the incorporation of the implementations of CCM and AEGIS into the modems, which would also involve an implementation of AES. An RBG would also be necessary to generate the nonces and IVs. Additionally, it would be necessary to automate the execution of these algorithms. For this, an intervention in the firmware of the modems would have to be done, which was not accessible to us, and would require a legal agreement with the EvoLogics manufacturer. Even the use of Janus with the modems requires a proprietary modification of Janus, which is not open source and was obtained through an agreement with NTNU. Hence, further modifications would be subject to additional legal procedures. Also, if the protocols are to be implemented on other modems from other manufacturers than EvoLogics, agreements would have to be made for these as well. Additionally, to perform in-water tests of the simulated protocols, it would be necessary to transport the modems to a marine environment, such as the Trondheim fjord, where they would need to perform the protocols repeatedly until a distance of 10 kilometers. This would require funding and it would pose a substantial organizational complexity.

6.2 Results

After simulating the protocols both with and without noise, and measuring the time needed for each version to complete in NS3, the round-trip times shown in Figure 6.3, Figure 6.4, Figure 6.5, and Figure 6.6, were obtained. Figure 6.3 shows the distance in meters between two devices performing authentication with respect to the time in milliseconds needed to perform the authentication, in the case without noise. The times for the protocols based on CCM and AEGIS both with and without cargo are shown. Additionally, the simulated time for the original protocol by Téglásy et al. is shown for comparison. In Figure 6.4 and Figure 6.5, the RTTs of the protocols with and without cargo, with varying levels of packet loss probability and varying distance, are shown for CCM and AEGIS, respectively. Figure 6.6 also shows the performance of the protocol by Téglásy et al. in a noisy channel, for comparison. The round-trip



(a) The modems were connected to a computer using a switch. A power source was used to provide power to the modems during longer periods of time.



(b) The modems had to be placed close to each other to exchange information in air.

Figure 6.2: The setup for in-air testing of the acoustic modems.

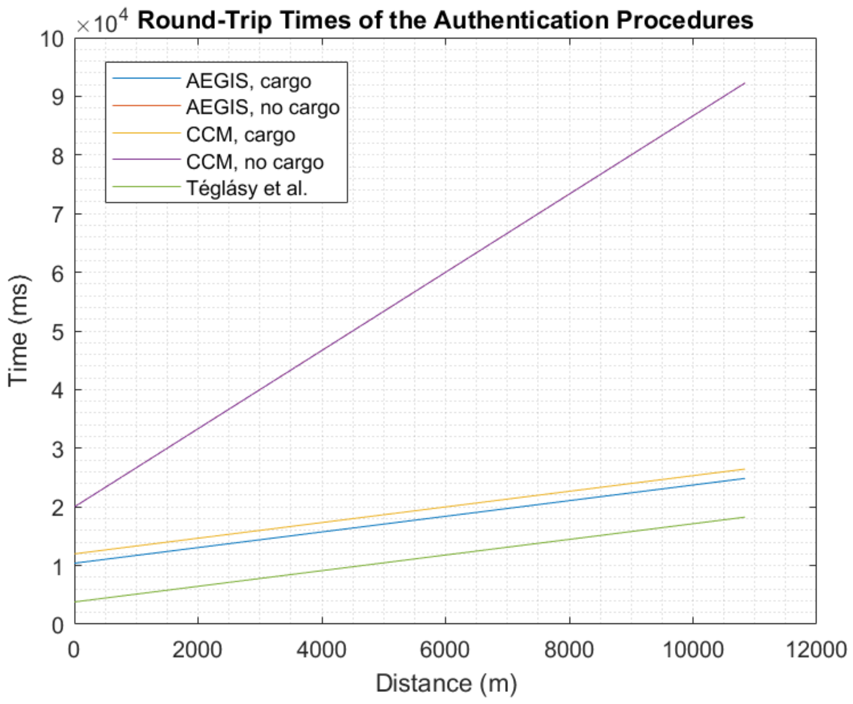


Figure 6.3: Round-trip times of the authentication protocols, using CCM and AEGIS, with and without cargo packets for both algorithms. The RC5-based solution by Téglásy et al. is included for comparison.

Table 6.3: The processing times of the security algorithms.

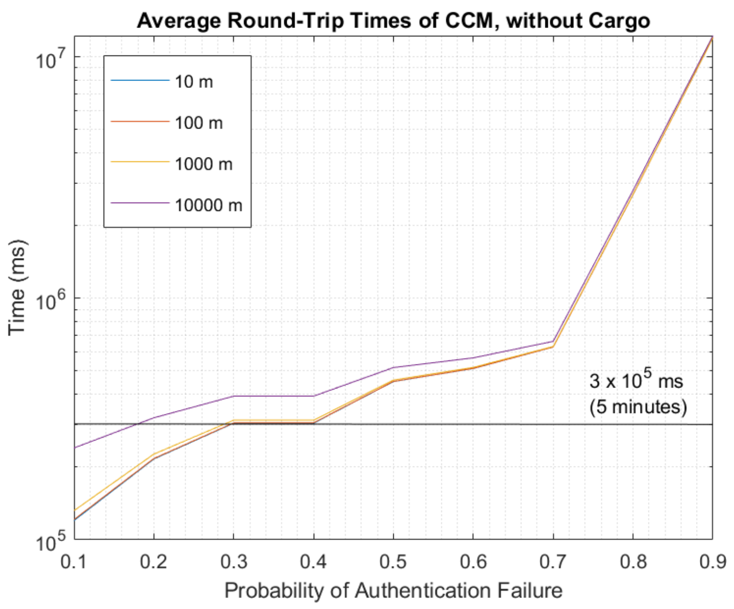
Algorithm	Generation-encryption	Decryption-verification
CCM	0.0160203 ms	0.0161610 ms
AEGIS	0.0368927 ms	0.0373279 ms
RC5[1]	0.002678 ms	0.0026934 ms

[1] RC5 only performs encryption and decryption and processes a smaller amount of data in our case.

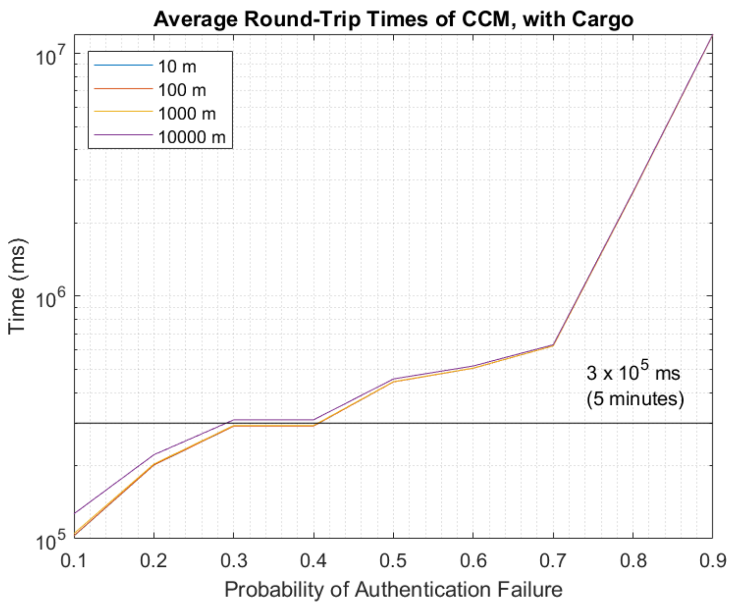
times are considered to be the total times needed to process, transmit, and receive all necessary data in both directions i.e., the sequence of events in Figure 6.1 performed twice. The times needed for the security algorithms to process their respective data are shown in Table 6.3.

6.3 Discussion

A discussion of the results presented in Section 6.2 follows.

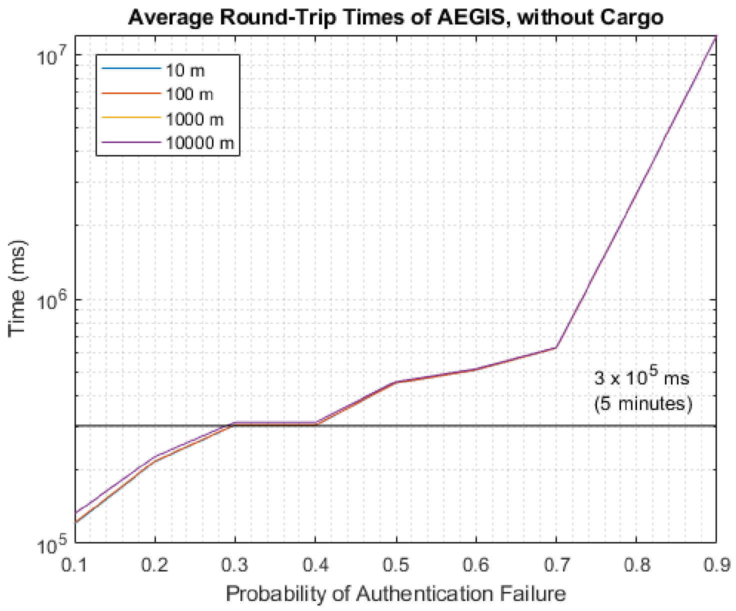


(a) RTTs of the CCM-based protocol without cargo in a noisy channel.

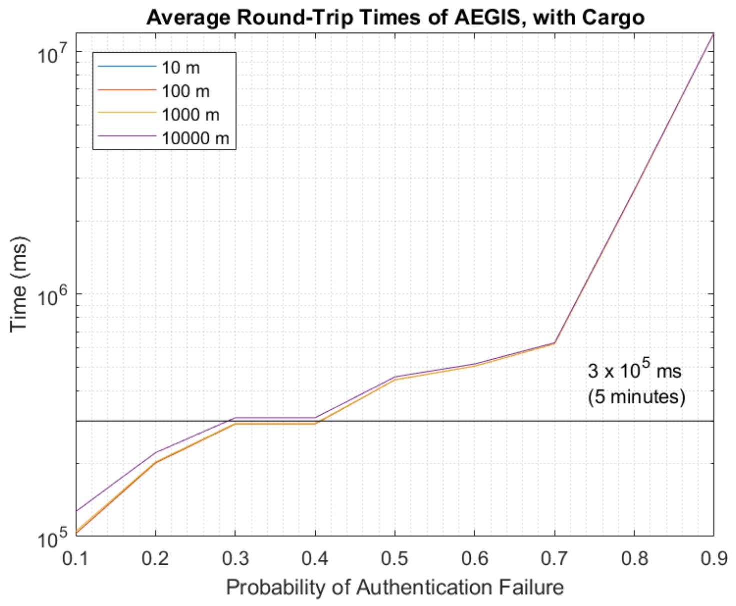


(b) RTTs of the CCM-based protocol with cargo in a noisy channel.

Figure 6.4: Comparison of RTTs for the CCM-based protocols with and without cargo, with varying levels of noise.



(a) RTTs of the AEGIS-based protocol without cargo in a noisy channel.



(b) RTTs of the AEGIS-based protocol with cargo in a noisy channel.

Figure 6.5: Comparison of RTTs for the AEGIS-based protocols with and without cargo, with varying levels of noise.

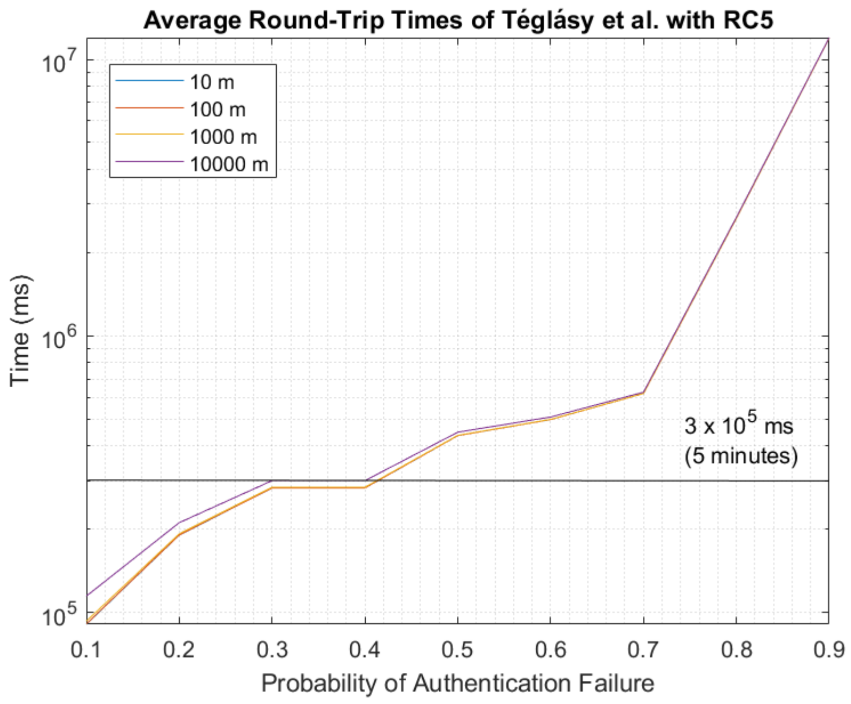


Figure 6.6: RTTs for the protocol by Téglásy et al., with varying levels of noise.

6.3.1 The Cases Without Noise

As expected, the times of the protocols with cargo in Figure 6.3 are substantially shorter than in the case of the cargo-less versions. This is due to the smaller amount of data to process and transmit, as there is only one Janus header and CRC transmitted in each direction. Moreover, the AEGIS-based protocol with cargo is marginally faster than its CCM-based counterpart, while the versions without cargo have practically identical times. Since the efficient usage of the AES round function and state variable of AEGIS has the purpose to increase its speed and performance, the expected result would be that the AEGIS-based protocols would be marginally faster. However, as shown in Table 6.3, in our simulations, CCM was the faster algorithm of the two. This may be due to our specific implementation, which was pedagogical and focused on readability rather than performance. Also, the repeated IV in AEGIS is longer than the repeated nonce in CCM (256 bits versus 64 bits, respectively), which also might have had an impact on the processing time. It is, however, worth noting that the processing times of all algorithms are negligible in comparison to the propagation time of the acoustic signals. Therefore, they have a minimal impact on the RTTs.

It was also expected that the original protocol by Téglásy et al. has the shortest round-trip time, as it only requires the transmission of one baseline packet in each direction and requires very little computation. However, the protocols presented in this thesis, especially the cargo versions, also have acceptable performance, with the shortest time being approximately 10 seconds for AEGIS with cargo when the devices are next to each other (0 meters). With the added security and scalability benefits, we believe this is an acceptable compromise.

Due to the simplistic Thorp propagation model used by NS3, the resulting curves in Figure 6.3 are linear. This is because no noise or packet loss is simulated in the communication channel, resulting in equal channel quality for all packets. However, a certain level of realism is still achieved, as the times shown are not the theoretical times that can be calculated for all algorithms and distances. For example, the theoretical time for the CCM-based protocol with cargo at a distance of 4000 m is $2 \cdot (0.0160203 \text{ ms} + 2 \cdot ((196 \text{ b}/80 \text{ bps}) \cdot 1000) + (4000 \text{ m}/1500 \text{ m/s}) \cdot 1000) + 0.0161610 \text{ ms}) = 15133.3977 \text{ ms}$. However, the time calculated with NS3 is 17332.064363 ms. Hence, some unpredictability is still provided. In addition, there are no results for distances beyond approximately 11000 m due to signal fading, which is an accurate portrayal of the behavior of Janus.

6.3.2 The Cases With Noise

When noise is added, it can be seen in both Figure 6.4 and Figure 6.5 that a longer propagation distance and higher packet loss probability contribute to higher RTTs. In this case, we defined an acceptable RTT of 5 minutes for the authentication procedure.

In Figure 6.4a it can be seen that, when cargo is not used, a distance of 10000 m causes the RTT to exceed 5 minutes at a packet loss probability of less than 0.2. When the distance is reduced, the RTT reaches the 5 minute limit at a higher packet loss probability i.e., around 0.3. On the other hand, when cargo is used, the RTT limit is reached after a probability of around 0.3 for all distances. This is expected, as the cargo versions are generally more reliable due to the smaller amount of data to transmit.

In the case of AEGIS, a longer distance again causes the 5 minute RTT limit to be reached sooner than with a shorter distance, which is expected. However, when cargo is not used, the limit at 10000 m is reached at a packet loss probability of around 0.3, compared to around 0.2 with CCM. When cargo is used, the results are practically identical with the CCM version.

With the protocol by Téglásy et al., the RTTs are shorter, which was expected, since only one baseline packet is transmitted in each direction and the RC5 processing is extremely fast. However, the average RTTs in Figure 6.6 are only marginally shorter than with CCM and AEGIS. The 5 minute time limit is reached at a failure probability of around 0.3 at 10000 m, while, at shorter distances, it is reached at a probability of just over 0.4. The reason for such a small difference from CCM and AEGIS is that, when noise is added, the noise is the biggest source of delay and not the propagation time of the signals. The delay added by the noise is much larger than the propagation time (at least 7 minutes, with one failure), making the propagation time small in comparison. Consequently, in a realistic mission scenario, where messages are transmitted in a noisy channel over longer periods of time, the performance of the new schemes based on CCM and AEGIS is very similar to that of the scheme by Téglásy et al.

To conclude, a higher packet loss probability will cause the authentication procedure to exceed the time limit sooner. As mentioned previously, packet loss can be caused by many factors. Deliberate DoS caused by, for instance, jamming (see Section 2.4.1), can provoke packet loss, but also channel quality fluctuations due to water currents, temperature, etc., can do this. Hence, if the desired maximum RTT is 5 minutes, the results presented above can serve as guidance in different mission conditions and network configurations. Note that, generally, the RTTs with noise are higher than without noise (with noise they are in the order of 10^5 to 10^7 ms, while, without noise, they are in the order of 10^4 ms.) This is expected, as each time a packet is lost and the authentication fails, 7 minutes (420000 ms) are added to the RTT.

More accurate measurements of the RTTs would be obtained if the protocols were realized on real acoustic modems. However, as discussed in Section 6.1.3, this was beyond the scope of this thesis.

Chapter 7

Conclusion and Future Work

In this thesis, we have proposed two AE schemes for providing confidentiality and integrity in wireless acoustic underwater communication using the Janus standard. We have incorporated these schemes into the first method for authentication between devices that use the Janus standard, defined by Téglásy et al. We have kept the original functionality of this protocol for ranging based on timestamps. We have shown that the new authentication procedures can be completed within an acceptable time period, in a simulation environment.

7.1 Answers to the Research Questions

Below is a short summary of the answers to the research questions of this thesis, defined in Section 1.3.

RQ1: How can authenticated encryption mechanisms be applied in Janus-based underwater communication to provide integrity and confidentiality of data?

- We have shown in Chapter 3 that our two identified mechanisms, CCM and AEGIS, can be adapted such that elements from Janus packets can be used as input to them. By performing these adaptations, we have realized the authentication protocol defined by Téglásy et al. using the mechanisms, both with and without Janus cargo packets.

RQ2: How can the underwater authenticated encryption schemes (ref. RQ1) be made to complete reliably within a required time?

- The selected security algorithms allow for the minimization of the communication overhead in the authentication protocols, which enables completion of the procedures in an acceptable time period. This was confirmed on the simulation level, as described in Chapter 6. The RTTs obtained satisfy our expectations and indicate the possibility of practical realization of the protocols.

RQ3: What level of security do the proposed schemes provide in relation to the underwater threat landscape?

- The underwater threat landscape, as well as the security requirements based on this landscape, were reviewed in Chapter 2. According to the STRIDE-based security analysis provided in Chapter 4, our proposed schemes are capable of resisting most attacks against the authentication protocols, except physical capture and DoS, from which no protocol or software algorithm can protect.

Based on the answers to the research questions that we obtained in this thesis, we can conclude that our proposed authentication methods, combined with the method proposed by Téglásy et al., have a potential for providing confidentiality and integrity of data in Janus-based wireless underwater communication. The research results obtained in this thesis were summarized in a paper, which was submitted to the Spanish national cryptologic conference, RECSI 2022[1].

7.2 Future Work

A major feature from Téglásy et al. that has been left out in this thesis is the generation and management of the session key K_{AB}, which was introduced in Section 2.8.1. The reason for this is that it was discovered during the work on the thesis that K_{AB} does not provide full forward secrecy in many cases and that the claim of forward secrecy relies on many assumptions. Hence, a project for the future is to provide symmetric authenticated key exchange with full forward secrecy, incorporated into the protocols defined in this thesis. For this, protocols based on key evolution and ratcheting can be used, such as those employed by the Signal protocol [Signal] or the ones proposed by Boyd et al. [BDdK+21].

Additionally, as discussed in Section 6.1.3, implementation of the protocols on real acoustic modems was beyond the scope of this thesis. Hence, an engineering project for the future would be to work towards agreements with manufacturers to allow for the implementation of the protocols on modems, such that modems could be sold as products with the support of the authentication protocols. An implementation of this kind would be done by implementing CCM and AEGIS with their respective dependencies into the modems and using standard Janus commands, such as those provided by the EvoLogics patch, to generate packets with the necessary user data for the protocols.

[1]The home page of the conference can be found here: https://recsi2022.unican.es/en/

References

[BDdK+21] C. Boyd, G. T. Davies, *et al.*, «Symmetric Key Exchange with Full Forward Security and Robust Synchronization», in *Advances in Cryptology – ASIACRYPT 2021*, M. Tibouchi and H. Wang, Eds., Cham: Springer International Publishing, 2021, pp. 681–710.

[BK98] A. Biryukov and E. Kushilevitz, «Improved cryptanalysis of RC5», in *Advances in Cryptology — EUROCRYPT'98*, K. Nyberg, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 85–99.

[Caesar] «Crypto competitions CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness». (), [Online]. Available: https://competitions.cr.yp.to/caesar.html (last visited: Jun. 15, 2022).

[CQP+05] C. Chaplin, E. Qi, *et al.*, «802.11i Overview», *IEEE 802.11-04*, Feb. 2005.

[DCT19] R. Diamant, P. Casari, and S. Tomasin, «Cooperative Authentication in Underwater Acoustic Sensor Networks», *IEEE Trans. Wirel. Commun.*, vol. 18, no. 2, pp. 954–968, 2019. [Online]. Available: https://doi.org/10.1109/TWC.2018.2886896.

[DEMS21] C. Dobraunig, M. Eichlseder, *et al.*, «Ascon v1.2: Lightweight Authenticated Encryption and Hashing», *J. Cryptol.*, vol. 34, no. 3, p. 33, 2021. [Online]. Available: https://doi.org/10.1007/s00145-021-09398-9.

[Dom11] M. C. Domingo, «Securing Underwater Wireless Communication Networks», *IEEE Wireless Communications*, vol. 18, no. 1, pp. 22–28, 2011.

[Dwo01] M. Dworkin, «Recommendation for Block Cipher Modes of Operation: Methods and Techniques», *NIST Special Publication 800-38A*, pp. 1–66, Dec. 2001.

[Dwo04] ——, «Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality», *NIST Special Publication 800-38C*, pp. 1–27, May 2004.

[Dwo05] ——, «Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication», *NIST Special Publication 800-38B*, pp. 1–21, May 2005.

[Dwo07] ——, «Recommendation for Block Cipher Modes of Operation: Galois/-Counter Mode (GCM) and GMAC», *NIST Special Publication 800-38D*, pp. 1–39, Nov. 2007.

[Evo18] EvoLogics, *EvoLogics Underwater Acoustic Positioning System User Guide*, Included with a set of EvoLogics acoustic modems., Feb. 2018.

[Fer02] N. Ferguson, «Michael: an improved MIC for 802.11 WEP», *IEEE P802.11 Wireless LANs*, Jan. 2002.

[Fer05] ——, «Authentication weakness in GCM», Jan. 2005.

[GB18] H. Ghannadrezaii and J.-F. Bousquet, «Securing a Janus-Based Flooding Routing Protocol for Underwater Acoustic Networks», in *OCEANS 2018 MTS/IEEE Charleston*, 2018, pp. 1–7.

[GitHub] «BranislavPetrovic/JanusSecurity: Source code of the simulations conducted for the thesis "Authentication and Encryption in Janus-Based Wireless Underwater Communications" at NTNU.» (), [Online]. Available: https://github.com/BranislavPetrovic/JanusSecurity.git (last visited: Jun. 19, 2022).

[HH21a] A.-M. Hobbs and S. Holdcroft, «Janus Class 17 "Venilia": Secure Pre-Canned Messaging», *Dstl Cyber and Information Systems*, pp. 1–22, May 2021, Requires account on the Janus Wiki.

[HH21b] ——, «Tiny Underwater Block cipher (TUBcipher): 27-bit Encryption Scheme for JANUS Class 17», *Dstl Cyber and Information Systems*, pp. 1–22, May 2021, Requires account on the Janus Wiki.

[HL15] R. Haakegaard and J. Lang, «The Elliptic Curve Diffe-Hellman (ECDH)», pp. 1–4, Dec. 2015.

[HM05] C. He and J. C. Mitchell, «Security analysis and improvements for IEEE 802.11i», in *In Proceedings of the 12th Annual Network and Distributed System Security Symposium*, 2005, pp. 90–110.

[JanusWiki] «JANUS Community Wiki | HomePage». (), [Online]. Available: http://www.januswiki.org/tiki-index.php (last visited: Jun. 15, 2022).

[Jea16] J. Jean, *TikZ for Cryptographers*, https://www.iacr.org/authors/tikz/, 2016.

[Jon03] J. Jonsson, «On the Security of CTR + CBC-MAC», in *Selected Areas in Cryptography*, K. Nyberg and H. Heys, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 76–93.

[LM12] J. Llor and M. P. Malumbres, «Underwater wireless sensor networks: How do acoustic propagation models impact the performance of higher-level protocols?», *Sensors (Basel)*, pp. 1313–1314, Jan. 2012.

[LSAR12] J. Lloret, S. Sendra, *et al.*, «Underwater Wireless Sensor Communications in the 2.4 GHz ISM Frequency Band», *Sensors*, vol. 12, no. 4, pp. 4237–4264, 2012. [Online]. Available: https://www.mdpi.com/1424-8220/12/4/4237.

[MV04] D. A. McGrew and J. Viega, «The Security and Performance of the Galois/-Counter Mode of Operation (Full Version)», *IACR Cryptol. ePrint Arch.*, p. 193, 2004. [Online]. Available: http://eprint.iacr.org/2004/193.

[NS3] «UAN Framework - Model Library». (), [Online]. Available: https://www.nsnam.org/docs/models/html/uan.html (last visited: Jun. 24, 2022).

[PAG+14] J. Potter, J. Alves, *et al.*, «The JANUS Underwater Communications Standard», *2014 Underwater Communications and Networking (UComms)*, pp. 1–4, Sep. 2014.

[PDLL16] C. Peng, X. Du, *et al.*, «An Ultra-Lightweight Encryption Scheme in Underwater Acoustic Networks», *Hindawi Publishing Corporation, Journal of Sensors*, vol. 2016, no. 8763528, pp. 1–10, Feb. 2016.

[PTB21] B. Petrovic, B. Z. Téglásy, and C. Boyd, *Securing Wireless Underwater Communications*, Nov. 2021.

[Riv95] R. L. Rivest, «The RC5 encryption algorithm», *Preneel B. (eds) Fast Software Encryption. FSE 1994*, pp. 86–96, 1995.

[RSG+00] P. Ryan, S. Schneider, *et al.*, *The Modelling and Analysis of Security Protocols: the CSP Approach.* Pearson Education, 2000, pp. 6–14.

[SCO+18] N. Shevchenko, T. A. Chick, *et al.*, «THREAT MODELING: A SUMMARY OF AVAILABLE METHODS», *Software Engineering Institute, Carnegie Mellon University*, pp. 1–2, Aug. 2018.

[Sha49] C. E. Shannon, «Communication Theory of Secrecy Systems», *Bell Systems Technical Journal*, no. 4, 1949.

[Signal] «Signal » Specifications » The Double Ratchet Algorithm». (), [Online]. Available: https://signal.org/docs/specifications/doubleratchet/ (last visited: Jun. 15, 2022).

[SKK10] P. Szalachowski, B. Ksiezopolski, and Z. Kotulski, «CMAC, CCM and GCM/GMAC: Advanced modes of operation of symmetric block ciphers in wireless sensor networks», *Information Processing Letters*, vol. 110, no. 7, pp. 247–251, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020019010000219.

[Sta17] W. Stallings, *Cryptography and Network Security, Principles and Practice.* Harlow: Pearson, 2017.

[TC91] T. I. Telegraph and T. C. Committee, «Security Architecture for Open Systems Interconnection for CCITT Applications, Recommendation X.800», *Data Communication Networks: Open Systems Interconnection (OSI); Security, Structure, and Applications*, pp. 8–9, Mar. 1991.

[TWPK] B. Z. Téglásy, E. Wengle, *et al.*, «Authentication of Underwater Assets», Paper in preparation.

[VP17] M. Vanhoef and F. Piessens, «Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2», in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17, Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1313–1328.

[VSPF21] Ø. Volden, P. Solnør, *et al.*, «Secure and Efficient Transmission of Vision-Based Feedback Control Signals», *Journal of Intelligent & Robotic Systems*, vol. 103, Sep. 2021.

[WHF02] D. Whiting, R. Housley, and N. Ferguson, «AES Encryption & Authentication Using CTR Mode & CBC-MAC», *IEEE 802.11-02*, pp. 1–16, Jan. 2002.

[Wie14] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Springer Verlag, 2014.

[WLLX19] J. Wang, C. Lu, *et al.*, «100 m/500 Mbps underwater optical wireless communication using an NRZ-OOK modulated 520 nm laser diode», *Opt. Express*, vol. 27, no. 9, pp. 12 171–12 181, Apr. 2019. [Online]. Available: http://opg.optica.org/oe/abstract.cfm?URI=oe-27-9-12171.

[WP13] H. Wu and B. Preneel, «AEGIS: A Fast Authenticated Encryption Algorithm», *Selected Areas in Cryptography (SAC 2013)*, pp. 1–21, 2013.

[Zap] G. Zappa, *README - JANUS Tool Kit 3.0.1*, http://www.januswiki.org/tiki-download_file.php?fileId=73, Requires account on the Janus Wiki. (last visited: Nov. 12, 2021).

Appendix A

TUBcipher

The TUBcipher is an SPN that utilizes a 256-bit key, 27-bit blocks, and 56 rounds of encryption. Each round of the cipher uses two subkeys, where one key is used in a keyed XOR stage and the other in a keyed permutation stage. The cipher also involves a linear diffusion stage and a fixed substitution stage. The structure of a round is shown in Figure A.1.

The 27-bit block size is chosen to provide confidentiality for the encrypted field in Figure 2.12. The remaining 7 bits of the ADB consist of a 5-bit IV and a 2-bit epoch identifier. Upon reception of a Venilia packet at the receiving device, the IV is left-padded with 0-bits until a length of 32 bits, before it is used in the TUBcipher. The epoch is also padded to the same length if it is too short. Although both the

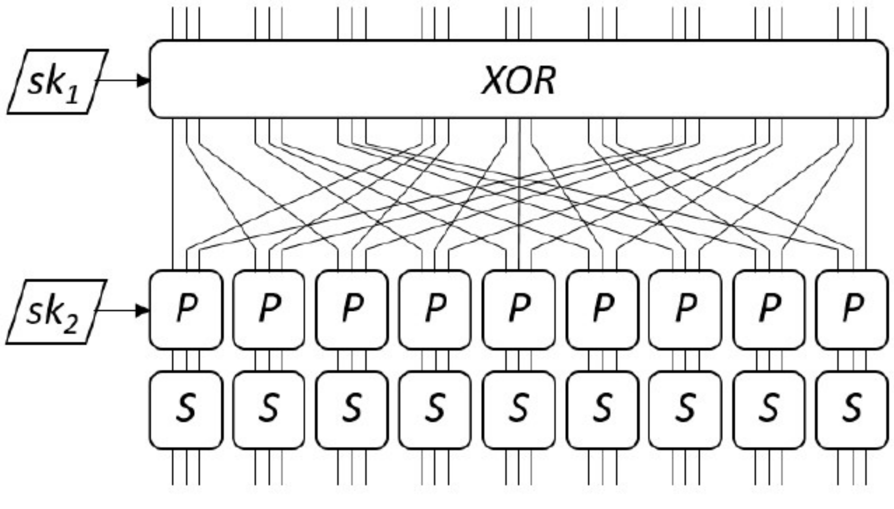


Figure A.1: A round of TUBcipher consisting of a keyed XOR, linear diffusion, keyed permutation, and fixed substitution. Source: [HH21b].

IV and the epoch are incremental counters, they are only used in the KDF of the key scheduler, and not in the encryption operation itself. The KDF produces the two subkeys sk_1 and sk_2 for all 56 rounds of the cipher, where sk_1 is 27 bits long, while sk_2 is 18 bits long. All subkeys are derived from a 2560-bit extended key K that is generated by concatenating the epoch, the key, the IV, and a 1-byte counter i, before calculating the digest of the 512-bit version of the Secure Hash Algorithm (SHA) of the resulting string five times, in the following way:

$$h_i = SHA512(epoch||key||IV||i), 1 \leq i \leq 5 \tag{A.1}$$

These digests are then concatenated together to produce the extended key K, from which sk_1 and sk_2 for all 56 rounds are derived:

$$K = h_0||h_1||h_2||h_3||h_4. \tag{A.2}$$

The IV provides for 32 independent ciphertexts per epoch. The epoch is a 32-bit variable and it indicates the time period in which the IV is required to be unique. The epoch duration must be known by all devices in a network, and the current epoch is identified by the two LSBs in the ADB. This allows for reuse of the 32 IVs, since the reuse of an IV under different epochs will result in a different ciphertext.

After the subkeys are created, the keyed XOR stage under sk_1 ensues. Since both the plaintext and sk_1 are 27 bits long, the plaintext bits are directly XORed with their respective key bits to produce an intermediate value, P_{XOR}. The operation is as follows:

$$P_{XOR}[i] = P[i] \oplus sk_1(r)[i], 0 \leq i \leq 26, \tag{A.3}$$

where r is the round number.

Following the XOR stage, a linear permutation takes place, as depicted in Figure A.1. The permutation is fixed and symmetric and requires no key. Each bit $P_{XOR}[i]$ is moved to the position $P_{DIFF}[j]$, where:

$$j = \begin{cases} 3i \bmod 26, & 0 \leq i \leq 25 \\ 26, & i = 26 \end{cases} \tag{A.4}$$

A keyed permutation then follows, in which the 27-bit block is split into 9 sub-blocks of 3 bits each, and the second 18-bit subkey sk_2 is split into 9 pairs of bits.

The permutation is performed according to a lookup table of bit pairs of the subkey and corresponding permutations of the 3-bit sub-blocks. Each bit pair of the subkey determines an order in which the sub-blocks should be permuted, resulting in a permutation that is dependent on the second subkey in each round.

Finally, the resulting 3-bit sub-blocks undergo a fixed substitution according to an S-box. The S-box simply determines which other sub-block each sub-block should be substituted with. After this substitution stage, a new round of TUBcipher begins. After the last round, the sub-blocks are concatenated together to produce the final 27-bit ciphertext.

Appendix B
Simulation Scripts

In this appendix, examples of simulation scripts that were used to conduct the experiments described in Section 6.1, are given. The examples include scripts from the Janus simulation in C, as well as protocol simulations in NS3, in C++. Only the CCM-based examples are shown i.e., the packet generation in Janus and the protocol in NS3, both with and without cargo. Only the portion of the script files that were modified in this project are shown. The complete source code that contains all simulations used for this thesis can be found in the project's GitHub repository [GitHub].

B.1 shows how five packets with example values were generated for the CCM-based protocol without cargo. The packets' binary representations were stored in their respective files for use in NS3. Similarly, B.2 shows the same procedure with one large cargo packet.

B.3 shows the protocol simulation of the CCM-based protocol without cargo, depicted in Figure 3.3. The appropriate processing delays for generation-encryption and decryption-verification, as well as the signal propagation delays with increasing distance, are added. In B.4, the same procedure for cargo packets is shown, depicted in Figure 3.4.

Source code B.1 The packet generation for the cargo-less CCM-based protocol, incorporated into the Janus simulation.

```
// --- CCM, no cargo ---
janus_packet_t ccmpkt1 = 0;
janus_packet_t ccmpkt2 = 0;
janus_packet_t ccmpkt3 = 0;
janus_packet_t ccmpkt4 = 0;
janus_packet_t ccmpkt5 = 0;
ccmpkt1 = janus_packet_new(params->verbose);
janus_uint8_t* ccmpkt1bytes = janus_packet_get_bytes(
ccmpkt1);
FILE* fp;
janus_uint8_t bin[JANUS_MIN_PKT_SIZE * 8];
clock_t encStart, encEnd;
double cpu_time_used;
double avgEnc, avgDec = 0;
int i;
janus_utils_dec2bin(ccmpkt1bytes, JANUS_MIN_PKT_SIZE, bin);
fp = fopen("/home/ttm4128/ccmnocargopkt1.txt", "wb");
uint8_t ccmpkt1adbbin[34];
uint8_t N[] = {0x00, 0x00, 0x00, 0x03};
uint8_t Ndouble[] = {0x00, 0x00, 0x00, 0x03, 0x00, 0x00,
0x00, 0x03};
uint8_t Nbin[32];
uint8_t key[] = {0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6,
0xc7, 0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf};
uint8_t A[3];
uint8_t Abin[24];
```

```
uint8_t n = 8;
ulong a = 3;
ulong p = 12;
uint8_t t = 4;
uint8_t err;
ulong c = p + t;
uint8_t* C = malloc(c * sizeof (uint8_t));
uint8_t cBin[8 * c];
uint8_t TSCD[] = {0x01, 0x02, 0x03, 0x04};
uint8_t MMSIA[] = {0x05, 0x06, 0x07, 0x08};
uint8_t MMSIB[] = {0x09, 0x10, 0x11, 0x12};
uint8_t MMSIAbin[32];
uint8_t MMSIBbin[32];
uint8_t P[12];

janus_utils_dec2bin(MMSIA, 4, MMSIAbin);
janus_utils_dec2bin(MMSIB, 4, MMSIBbin);
MMSIAbin[30] = MMSIAbin[31] = 0;
MMSIBbin[30] = MMSIBbin[31] = 0;
janus_utils_bin2dec(MMSIAbin, 32, MMSIA);
janus_utils_bin2dec(MMSIBbin, 32, MMSIB);

for (i = 0; i < 4; i++) {
    P[i] = TSCD[i];
    P[i + 4] = MMSIA[i];
    P[i + 8] = MMSIB[i];
}

for (i = 0; i < 64; i++) {
    if (i >= 0 && i <= 21) {
        Abin[i] = bin[i];
    } else if (i >= 54 && i <= 55) {
        Abin[i - 32] = bin[i];
    }
}
```

```
        janus_utils_bin2dec(Abin, 24, A);


        err = CCM_gen_encrypt(key, Ndouble, n, A, a, P, p, t, C, c);

        // --- packet 1 ---
        janus_utils_dec2bin(N, 4, Nbin);

        for (i = 0; i < 32; i++) {
            bin[i + 22] = Nbin[i];
        }

        janus_utils_bin2dec(bin, 64, ccmpkt1bytes);
        janus_packet_set_bytes(ccmpkt1, ccmpkt1bytes);

        // Initialize state.
        state = janus_tx_state_new((params->verbose > 1));

        // Transmit.
        janus_simple_tx_execute(simple_tx, ccmpkt1, state);

        janus_utils_dec2bin(ccmpkt1bytes, JANUS_MIN_PKT_SIZE, bin);

        for (i = 0; i < 64; i++)
            fprintf(fp, "%u", bin[i]);
        fclose(fp);

        // --- packet 2 ---
        fp = fopen("/home/ttm4128/ccmnocargopkt2.txt", "wb");
        janus_utils_dec2bin(C, 16, cBin);

        ccmpkt2 = janus_packet_new(params->verbose);
        janus_uint8_t* ccmpkt2bytes = janus_packet_get_bytes(
        ccmpkt2);
        janus_utils_dec2bin(ccmpkt2bytes, JANUS_MIN_PKT_SIZE, bin);

        for (i = 0; i < 32; i++) {
            bin[i + 22] = cBin[i];
        }

        janus_utils_bin2dec(bin, 64, ccmpkt2bytes);
        janus_packet_set_bytes(ccmpkt2, ccmpkt2bytes);
```

```
// Initialize state.
state = janus_tx_state_new((params->verbose > 1));

// Transmit.
janus_simple_tx_execute(simple_tx, ccmpkt2, state);

janus_utils_dec2bin(ccmpkt2bytes, JANUS_MIN_PKT_SIZE, bin);

for (i = 0; i < 64; i++)
    fprintf(fp, "%u", bin[i]);
fclose(fp);

// --- packet 3 ---
fp = fopen("/home/ttm4128/ccmnocargopkt3.txt", "wb");
ccmpkt3 = janus_packet_new(params->verbose);
janus_uint8_t* ccmpkt3bytes = janus_packet_get_bytes(
ccmpkt3);
janus_utils_dec2bin(ccmpkt3bytes, JANUS_MIN_PKT_SIZE, bin);

for (i = 0; i < 32; i++) {
    bin[i + 22] = cBin[i + 32];
}

janus_utils_bin2dec(bin, 64, ccmpkt3bytes);
janus_packet_set_bytes(ccmpkt3, ccmpkt3bytes);

// Initialize state.
state = janus_tx_state_new((params->verbose > 1));

// Transmit.
janus_simple_tx_execute(simple_tx, ccmpkt3, state);

janus_utils_dec2bin(ccmpkt3bytes, JANUS_MIN_PKT_SIZE, bin);

for (i = 0; i < 64; i++)
    fprintf(fp, "%u", bin[i]);
fclose(fp);
```

```
// --- packet 4 ---
fp = fopen("/home/ttm4128/ccmnocargopkt4.txt", "wb");
ccmpkt4 = janus_packet_new(params->verbose);
janus_uint8_t* ccmpkt4bytes = janus_packet_get_bytes(
ccmpkt4);
janus_utils_dec2bin(ccmpkt4bytes, JANUS_MIN_PKT_SIZE, bin);

for (i = 0; i < 32; i++) {
    bin[i + 22] = cBin[i + 64];
}

janus_utils_bin2dec(bin, 64, ccmpkt4bytes);
janus_packet_set_bytes(ccmpkt4, ccmpkt4bytes);

// Initialize state.
state = janus_tx_state_new((params->verbose > 1));

// Transmit.
janus_simple_tx_execute(simple_tx, ccmpkt4, state);

janus_utils_dec2bin(ccmpkt4bytes, JANUS_MIN_PKT_SIZE, bin);

for (i = 0; i < 64; i++)
    fprintf(fp, "%u", bin[i]);
fclose(fp);

// --- packet 5 ---
fp = fopen("/home/ttm4128/ccmnocargopkt5.txt", "wb");
ccmpkt5 = janus_packet_new(params->verbose);
janus_uint8_t* ccmpkt5bytes = janus_packet_get_bytes(
ccmpkt5);
janus_utils_dec2bin(ccmpkt5bytes, JANUS_MIN_PKT_SIZE, bin);

for (i = 0; i < 32; i++) {
    bin[i + 22] = cBin[i + 96];
}

janus_utils_bin2dec(bin, 64, ccmpkt5bytes);
janus_packet_set_bytes(ccmpkt5, ccmpkt5bytes);

// Initialize state.
state = janus_tx_state_new((params->verbose > 1));
```

```
    // Transmit.
    janus_simple_tx_execute(simple_tx, ccmpkt5, state);

    janus_utils_dec2bin(ccmpkt5bytes, JANUS_MIN_PKT_SIZE, bin);

    for (i = 0; i < 64; i++)
        fprintf(fp, "%u", bin[i]);
    fclose(fp);
```

Source code B.2 The packet generation for the CCM-based protocol with cargo, incorporated into the Janus simulation.

```
// --- CCM, cargo ---
janus_packet_t pkt = 0;
pkt = janus_packet_new(params->verbose);
janus_uint8_t* pktbytes = janus_packet_get_bytes(pkt);
FILE* fp;
janus_uint8_t bin[JANUS_MIN_PKT_SIZE * 8];
clock_t encStart, encEnd;
double cpu_time_used;
double avgEnc, avgDec = 0;
int i;
janus_utils_dec2bin(pktbytes, JANUS_MIN_PKT_SIZE, bin);
fp = fopen("/home/ttm4128/ccmcargopkt.txt", "wb");
uint8_t pktadbbin[34];
uint8_t N[] = {0x00, 0x00, 0x00, 0x03};
uint8_t Ndouble[] = {0x00, 0x00, 0x00, 0x03, 0x00,
0x00, 0x00, 0x03};
uint8_t Nbin[32];
uint8_t key[] = {0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6,
0xc7, 0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf};
uint8_t F[] = {0x02};
uint8_t Fbin[8];
uint8_t A[3];
uint8_t Abin[24];
uint8_t n = 8;
ulong a = 3;
ulong p = 12;
uint8_t t = 4;
uint8_t err;
ulong c = p + t;
uint8_t* C = malloc(c * sizeof (uint8_t));
uint8_t cBin[8 * c];
uint8_t TSCD[] = {0x01, 0x02, 0x03, 0x04};
uint8_t MMSIA[] = {0x05, 0x06, 0x07, 0x08};
uint8_t MMSIB[] = {0x09, 0x10, 0x11, 0x12};
uint8_t MMSIAbin[32];
uint8_t MMSIBbin[32];
uint8_t P[12];
```

```
janus_utils_dec2bin(MMSIA, 4, MMSIAbin);
janus_utils_dec2bin(MMSIB, 4, MMSIBbin);
MMSIAbin[30] = MMSIAbin[31] = 0;
MMSIBbin[30] = MMSIBbin[31] = 0;
janus_utils_bin2dec(MMSIAbin, 32, MMSIA);
janus_utils_bin2dec(MMSIBbin, 32, MMSIB);

for (i = 0; i < 4; i++) {
    P[i] = TSCD[i];
    P[i + 4] = MMSIA[i];
    P[i + 8] = MMSIB[i];
}

for (i = 0; i < 64; i++) {
    if (i >= 0 && i <= 21) {
        Abin[i] = bin[i];
    } else if (i >= 54 && i <= 55) {
        Abin[i - 32] = bin[i];
    }
}

janus_utils_bin2dec(Abin, 24, A);

err = CCM_gen_encrypt(key, Ndouble, n, A, a, P, p, t, C, c);

// --- packet ---
janus_utils_dec2bin(N, 4, Nbin);

for (i = 0; i < 26; i++) {
    bin[i + 30] = Nbin[i];
}

janus_utils_bin2dec(bin, 64, pktbytes);
janus_packet_set_bytes(pkt, pktbytes);

// Initialize state.
state = janus_tx_state_new((params->verbose > 1));

// Transmit.
janus_simple_tx_execute(simple_tx, pkt, state);

janus_utils_dec2bin(pktbytes, JANUS_MIN_PKT_SIZE, bin);
```

```
        for (i = 0; i < 64; i++) {
            fprintf(fp, "%u", bin[i]);
        }

        for (i = 0; i < 6; i++) {
            fprintf(fp, "%u", Nbin[i + 26]);
        }

        janus_utils_dec2bin(F, 1, Fbin);

        fprintf(fp, "%u", Fbin[6]);
        fprintf(fp, "%u", Fbin[7]);

        janus_utils_dec2bin(C, 16, cBin);

        for (i = 0; i < 124; i++) {
            fprintf(fp, "%u", cBin[i]);
        }

        fclose(fp);
```

Source code B.3 The protocol simulation of the cargo-less CCM-based protocol in NS3.

```
private:
    NodeContainer m_nodes; //!< UAN nodes
    std::map<Ptr<Node>, Ptr<Socket> > m_sockets; //!< send and
    receive sockets
    double timestamp;
    double oneWay;
    double rtt;
    ulong protI = 0;
    int pktNum;
};

void
UanExperiment::SetupPositions() {
    MobilityHelper mobilityHelper;
    mobilityHelper.SetMobilityModel(
    "ns3::ConstantPositionMobilityModel");
    mobilityHelper.Install(m_nodes);
    m_nodes.Get(0)->GetObject<MobilityModel> ()->
    SetPosition(Vector(0, 0, 0));
    m_nodes.Get(1)->GetObject<MobilityModel> ()->
    SetPosition(Vector(1, 0, 0));
}

void
UanExperiment::SetupCommunications() {
    Ptr<UanChannel> channel = CreateObject<UanChannel> ();
    //Ptr<UanPropModel> prop = CreateObject<UanPropModelIdeal>();
    Ptr<UanPropModel> prop = CreateObject<UanPropModelThorp>();
    channel->SetPropagationModel(prop);
    UanHelper uanHelper;
    NetDeviceContainer netDeviceContainer = uanHelper.Install(
    m_nodes, channel);
}
```

```
void
UanExperiment::PrintReceivedPacket(Ptr<Socket> socket) {
    Address srcAddress;
    double distance = m_nodes.Get (1)->
    GetObject<MobilityModel> ()->GetPosition ().x;
    while (socket->GetRxAvailable() > 0) {
        Ptr<Packet> packet1 = socket->RecvFrom(srcAddress);
        PacketSocketAddress packetSocketAddress5 =
        PacketSocketAddress::ConvertFrom(srcAddress);
        srcAddress = packetSocketAddress5.GetPhysicalAddress();

        uint32_t k;
        int l;
        double a, b, c, d;
        string strpkt1, strpkt2, strpkt3, strpkt4, strpkt5;
        ifstream in1("/home/ttm4128/ccmnocargopkt1.txt", ios_base::in);
        ifstream in2("/home/ttm4128/ccmnocargopkt2.txt", ios_base::in);
        ifstream in3("/home/ttm4128/ccmnocargopkt3.txt", ios_base::in);
        ifstream in4("/home/ttm4128/ccmnocargopkt4.txt", ios_base::in);
        ifstream in5("/home/ttm4128/ccmnocargopkt5.txt", ios_base::in);

        //NS_LOG_UNCOND(pktNum);
        if (pktNum == 1) {
            in1 >> strpkt1;
            //NS_LOG_UNCOND(strpkt1);
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            c = ((double) l / 80.0) * 1000; // janus RX
            d = ((double) l / 80.0) * 1000 + 0.0160203; // janus TX +
            generation-encryption

            timestamp = a + c + d;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket2To1, this);
        } else if (pktNum == 2) {
            in2 >> strpkt1;
            //NS_LOG_UNCOND(strpkt1);
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            c = ((double) l / 80.0) * 1000; // janus RX
            d = ((double) l / 80.0) * 1000; // janus TX +
            generation-encryption
            timestamp = a + c + d;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket3To1, this);
        } else if (pktNum == 3) {
            in3 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            c = 5 * ((double) l / 80.0) * 1000; // janus RX
            d = 5 * ((double) l / 80.0) * 1000;
            // janus TX + generation-encryption

            timestamp = a + c + d;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket4To1, this);
        } else if (pktNum == 4) {
            in4 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            c = 5 * ((double) l / 80.0) * 1000; // janus RX
            d = 5 * ((double) l / 80.0) * 1000; // janus TX +
            generation-encryption

            timestamp = a + c + d;
            oneWay = timestamp - oneWay;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket5To1, this);
        } else if (pktNum == 5) {
            in1 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            b = 0.0161610; // decryption-verification
            c = 5 * ((double) l / 80.0) * 1000; // janus RX
            d = 5 * ((double) l / 80.0) * 1000 + 0.0160203;
            // janus TX + generation-encryption

            timestamp = a + b + c + d;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket1To0, this);
        }
```

```
          else if (pktNum == 6) {
            in2 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            c = 5 * ((double) l / 80.0) * 1000; // janus RX
            d = 5 * ((double) l / 80.0) * 1000;
            // janus TX + generation-encryption

            timestamp = a + c + d;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket2To0, this);
        } else if (pktNum == 7) {
            in3 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            c = 5 * ((double) l / 80.0) * 1000; // janus RX
            d = 5 * ((double) l / 80.0) * 1000;
            // janus TX + generation-encryption

            timestamp = a + c + d;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket3To0, this);
        } else if (pktNum == 8) {
            in3 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            c = 5 * ((double) l / 80.0) * 1000; // janus RX
            d = 5 * ((double) l / 80.0) * 1000;
            // janus TX + generation-encryption

            timestamp = a + c + d;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket4To0, this);
        } else if (pktNum == 9) {
            in3 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            c = 5 * ((double) l / 80.0) * 1000; // janus RX
            d = 5 * ((double) l / 80.0) * 1000;
            // janus TX + generation-encryption
            timestamp = a + c + d;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacket5To0, this);
        } else if (pktNum == 10) {
            in4 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            b = 0.0161610; // decryption-verification
            c = 5 * ((double) l / 80.0) * 1000; // janus RX
            d = 5 * ((double) l / 80.0) * 1000 + 0.0160203;
            // janus TX + generation-encryption

            timestamp = a + b + c + d;
            rtt = timestamp - rtt;
            NS_LOG_UNCOND(to_string(distance) << ", " <<
            to_string(rtt));
            distance += 10;
            m_nodes.Get(1)->GetObject<MobilityModel> ()->
            SetPosition(Vector(distance, 0, 0));

            if (protI == 1) {
                k = timestamp;
            }
            Simulator::Schedule(MilliSeconds(420000 - k),
            &UanExperiment::SendPacket1To1, this);
        }

        in1.close();
        in2.close();
        in3.close();
        in4.close();
        in5.close();

        while (l % 8 != 0) {
            l++;
        }

        uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 *
        sizeof (uint8_t));
        string binPacket1, binPacket2, binPacket3, binPacket4;
```

```
        packet1->CopyData(janusDecArr1, l / 8);

        for (int i = 0; i < l / 8; i++) {
            binPacket1 += bitset<8>(*(janusDecArr1 + i)).to_string();
        }
//        NS_LOG_UNCOND(binPacket1);
        free(janusDecArr1);
    }
}

int chartobin(char c) {
    if (c == '1')
        return 1;
    else
        return 0;
}

void UanExperiment::SendPacket1To1() {
    NodeContainer::Iterator node = m_nodes.Begin();
    node++;
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node = m_nodes.Begin();
    protI++;
    pktNum = 1;
    string strpkt1;
    ifstream in1("/home/ttm4128/ccmnocargopkt1.txt", ios_base::in);
    in1 >> strpkt1;
    int l = strpkt1.length();

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;
    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt1[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr1 + janusIndex) = (uint8_t) b;
        }
    }

    in1.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr1, l / 8);
    delete janusDecArr1;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    rtt = (double)(Simulator::Now().GetMilliSeconds());
    timestamp = oneWay = (double)Simulator::Now().GetMilliSeconds() +
    0.0160203 + ((double)l / 80.0) * 1000;
    // Now + generation-encryption + Janus encoding
    //NS_LOG_UNCOND("Node 0 | sent 1 packet to 1, timestamp: " <<
    to_string(timestamp));
}

void UanExperiment::SendPacket2To1() {
    NodeContainer::Iterator node = m_nodes.Begin();
    node++;
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node = m_nodes.Begin();
    pktNum = 2;

    string strpkt;
    ifstream in1("/home/ttm4128/ccmnocargopkt2.txt", ios_base::in);
    in1 >> strpkt;
    int l = strpkt.length();
    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr1 + janusIndex) = (uint8_t) b;
        }
    }

    in1.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr1, l / 8);
    delete janusDecArr1;

    Simulator::Schedule(Seconds(1), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);

    timestamp += ((double)l / 80.0) * 1000; // Now + Janus encoding
    //NS_LOG_UNCOND("Node 0 | sent 2 packets to 1, timestamp: " <<
    to_string(timestamp));
}

void UanExperiment::SendPacket3To1() {
    NodeContainer::Iterator node = m_nodes.Begin();
    node++;
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node = m_nodes.Begin();
    pktNum = 3;
    string strpkt;
    ifstream in("/home/ttm4128/ccmnocargopkt3.txt", ios_base::in);
    in >> strpkt;
    int l = strpkt.length();

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr + janusIndex) = (uint8_t) b;
        }
    }

    in.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr, l / 8);
    delete janusDecArr;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    timestamp += ((double)l / 80.0) * 1000; // Now + Janus encoding
    //NS_LOG_UNCOND("Node 0 | sent 3 packets to 1, timestamp: " <<
    to_string(timestamp));
}
```

```
void UanExperiment::SendPacket4To1() {
    NodeContainer::Iterator node = m_nodes.Begin();
    node++;
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node = m_nodes.Begin();
    pktNum = 4;

    string strpkt;
    ifstream in("/home/ttm4128/ccmnocargopkt4.txt", ios_base::in);
    in >> strpkt;
    int l = strpkt.length();

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr + janusIndex) = (uint8_t) b;
        }
    }

    in.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr, l / 8);
    delete janusDecArr;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    timestamp += ((double)l / 80.0) * 1000; // Now + Janus encoding
    //NS_LOG_UNCOND("Node 0 | sent 4 packets to 1, timestamp: " <<
    to_string(timestamp));
}

void UanExperiment::SendPacket5To1() {
    NodeContainer::Iterator node = m_nodes.Begin();
    node++;
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node = m_nodes.Begin();
    pktNum = 5;

    string strpkt;
    ifstream in("/home/ttm4128/ccmnocargopkt5.txt", ios_base::in);
    in >> strpkt;
    int l = strpkt.length();

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr + janusIndex) = (uint8_t) b;
        }
    }

    in.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr, l / 8);
    delete janusDecArr;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    timestamp += ((double)l / 80.0) * 1000; // Now + Janus encoding
    //NS_LOG_UNCOND("Node 0 | sent 5 packets to 1, timestamp: " <<
    to_string(timestamp));
}

void UanExperiment::SendPacket1To0() {
    NodeContainer::Iterator node = m_nodes.Begin();
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node++;

    string strpkt1;
    ifstream in1("/home/ttm4128/ccmnocargopkt1.txt", ios_base::in);
    in1 >> strpkt1;
    int l = strpkt1.length();
    pktNum = 6;

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt1[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr1 + janusIndex) = (uint8_t) b;
        }
    }

    in1.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr1, l / 8);
    delete janusDecArr1;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    timestamp += 0.0161610 + ((double)l / 80.0) * 1000;
    // Now + generation-encryption + Janus encoding
    //NS_LOG_UNCOND("Node 1 | sent 1 packet to 0, timestamp: " <<
    to_string(timestamp));
}

void UanExperiment::SendPacket2To0() {
    NodeContainer::Iterator node = m_nodes.Begin();
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node++;

    string strpkt1;
    ifstream in1("/home/ttm4128/ccmnocargopkt2.txt", ios_base::in);
    in1 >> strpkt1;
    int l = strpkt1.length();
    pktNum = 7;

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt1[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr1 + janusIndex) = (uint8_t) b;
        }
    }

    in1.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr1, l / 8);
    delete janusDecArr1;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    timestamp += ((double)l / 80.0) * 1000; // Janus encoding
    //NS_LOG_UNCOND("Node 1 | sent 2 packets to 0, timestamp: "
    << to_string(timestamp));
}

void UanExperiment::SendPacket3To0() {
    NodeContainer::Iterator node = m_nodes.Begin();
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node++;

    string strpkt1;
    ifstream in1("/home/ttm4128/ccmnocargopkt3.txt", ios_base::in);
    in1 >> strpkt1;
    int l = strpkt1.length();
    pktNum = 8;

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt1[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr1 + janusIndex) = (uint8_t) b;
        }
    }

    in1.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr1, l / 8);
    delete janusDecArr1;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    timestamp += ((double)l / 80.0) * 1000; // Janus encoding
    //NS_LOG_UNCOND("Node 1 | sent 3 packets to 0, timestamp: " <<
    to_string(timestamp));
}

void UanExperiment::SendPacket4To0() {
    NodeContainer::Iterator node = m_nodes.Begin();
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node++;

    string strpkt1;
    ifstream in1("/home/ttm4128/ccmnocargopkt4.txt", ios_base::in);
    in1 >> strpkt1;
    int l = strpkt1.length();
    pktNum = 9;

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt1[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr1 + janusIndex) = (uint8_t) b;
        }
    }

    in1.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr1, l / 8);
    delete janusDecArr1;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    timestamp += ((double)l / 80.0) * 1000; // Janus encoding
    //NS_LOG_UNCOND("Node 1 | sent 4 packets to 0, timestamp: " <<
    to_string(timestamp));
}

void UanExperiment::SendPacket5To0() {
    NodeContainer::Iterator node = m_nodes.Begin();
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node++;

    string strpkt1;
    ifstream in1("/home/ttm4128/ccmnocargopkt5.txt", ios_base::in);
    in1 >> strpkt1;
    int l = strpkt1.length();
    pktNum = 10;

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;
    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = strpkt1[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr1 + janusIndex) = (uint8_t) b;
        }
    }

    in1.close();
    Ptr<Packet> pkt1 = Create<Packet> (janusDecArr1, l / 8);
    delete janusDecArr1;

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt1, dst);
    timestamp += ((double)l / 80.0) * 1000; // Janus encoding
    //NS_LOG_UNCOND("Node 1 | sent 5 packets to 0, timestamp: " <<
    to_string(timestamp));
}

void
UanExperiment::SendSinglePacket(Ptr<Node> node, Ptr<Packet> pkt,
Mac8Address dst) {
    //NS_LOG_UNCOND ( Simulator::Now ().GetHours () << "h" <<
    " packet sent to " << dst );
    PacketSocketAddress socketAddress;
    socketAddress.SetSingleDevice(node->GetDevice(0)->GetIfIndex());
    socketAddress.SetPhysicalAddress(dst);
    socketAddress.SetProtocol(0);
    m_sockets[node]->SendTo(pkt, 0, socketAddress);
}
```

```
void
UanExperiment::Prepare() {
    m_nodes.Create(2);
    SetupPositions();
    SetupCommunications();
    SetupApplications();
    SendPacket1To1();
}

int
main(int argc, char *argv[]) {
    CommandLine cmd(__FILE__);
    cmd.Parse(argc, argv);

    UanExperiment experiment;
    experiment.Prepare();

    Simulator::Stop(Days(6));
    Simulator::Run();
    Simulator::Destroy();

    experiment.Teardown();

    return 0;
}
```

Source code B.4 The protocol simulation of the CCM-based protocol with cargo, in NS3.

```
private:
    NodeContainer m_nodes; //!< UAN nodes
    std::map<Ptr<Node>, Ptr<Socket> > m_sockets;
    //!< send and receive sockets
    double timestamp;
    double oneWay;
    double rtt;
    ulong protI = 0;
    int pktNum;
};

void
UanExperiment::SetupPositions() {
    MobilityHelper mobilityHelper;
    mobilityHelper.SetMobilityModel("ns3::
    ConstantPositionMobilityModel");
    mobilityHelper.Install(m_nodes);
    m_nodes.Get(0)->GetObject<MobilityModel> ()->
    SetPosition(Vector(0, 0, 0));
    m_nodes.Get(1)->GetObject<MobilityModel> ()->
    SetPosition(Vector(1, 0, 0));
}

void
UanExperiment::SetupCommunications() {
    Ptr<UanChannel> channel = CreateObject<UanChannel> ();
    //Ptr<UanPropModel> prop = CreateObject<UanPropModelIdeal>();
    Ptr<UanPropModel> prop = CreateObject<UanPropModelThorp>();
    channel->SetPropagationModel(prop);
    UanHelper uanHelper;
    NetDeviceContainer netDeviceContainer = uanHelper.Install(
    m_nodes, channel);
}
```

```
void
UanExperiment::PrintReceivedPacket(Ptr<Socket> socket) {
    Address srcAddress;
    double distance = m_nodes.Get (1)->GetObject<MobilityModel> ()->
    GetPosition ().x;
    while (socket->GetRxAvailable() > 0) {
        Ptr<Packet> packet1 = socket->RecvFrom(srcAddress);
        PacketSocketAddress packetSocketAddress5 =
        PacketSocketAddress::ConvertFrom(srcAddress);
        srcAddress = packetSocketAddress5.GetPhysicalAddress();
        uint32_t k;
        int l;
        double a, b, c, d;
        string strpkt1;
        ifstream in1("/home/ttm4128/ccmcargopkt.txt", ios_base::in);

        if(Mac8Address::ConvertFrom(srcAddress) == 0) {
            in1 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            b = 0.0161610; // decryption-verification
            c = ((double) l / 80.0) * 1000; // janus RX
            d = ((double) l / 80.0) * 1000 + 0.0160203;
            // janus TX + generation-encryption

            timestamp = a + b + c + d;
            oneWay = timestamp - oneWay;

            Simulator::Schedule(Seconds(0), &UanExperiment::
            SendPacketTo0, this);
        } else {
            in1 >> strpkt1;
            l = strpkt1.length();
            a = (double) Simulator::Now().GetMilliSeconds();
            b = 0.0161610; // decryption-verification
            c = ((double) l / 80.0) * 1000; // janus RX
            d = ((double) l / 80.0) * 1000 + 0.0160203;
            // janus TX + generation-encryption

            timestamp = a + b + c + d;
            rtt = timestamp - rtt;
            NS_LOG_UNCOND(to_string(distance) << ", " <<
            to_string(rtt));
            distance += 10;
            m_nodes.Get(1)->GetObject<MobilityModel> ()->
            SetPosition(Vector(distance, 0, 0));

            if (protI == 1) {
                k = timestamp;
            }
            Simulator::Schedule(MilliSeconds(420000 - k),
            &UanExperiment::SendPacketTo1, this);
        }

        in1.close();

        while (l % 8 != 0) {
            l++;
        }

        uint8_t* janusDecArr1 = (uint8_t*) malloc(l / 8 * sizeof (
        uint8_t));
        string binPacket1;

        packet1->CopyData(janusDecArr1, l / 8);

        for (int i = 0; i < l / 8; i++) {
            binPacket1 += bitset<8>(*(janusDecArr1 + i)).to_string();
        }
//        NS_LOG_UNCOND(binPacket1);
        free(janusDecArr1);
    }
}

int chartobin(char c) {
    if (c == '1')
        return 1;
    else
        return 0;
}
```

```
void UanExperiment::SendPacketTo1() {
    NodeContainer::Iterator node = m_nodes.Begin();
    node++;
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node = m_nodes.Begin();
    protI++;
    string packet;
    ifstream in("/home/ttm4128/ccmcargopkt.txt", ios_base::in);
    in >> packet;
    int l = packet.length();
    double numL = l;

    while (l % 8 != 0) {
        l++;
    }

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;

    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = packet[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr + janusIndex) = (uint8_t) b;
        }
    }

    in.close();
    Ptr<Packet> pkt = Create<Packet> (janusDecArr, l / 8);

    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt, dst);
    rtt = (double)(Simulator::Now().GetMilliSeconds());
    timestamp = oneWay = (double)(Simulator::Now().GetMilliSeconds())
    + 0.0160203 + (numL / 80.0) * 1000; // Now + enc + Janus encoding
    //NS_LOG_UNCOND("Sent packet to 1, timestamp: " <<
    to_string(timestamp) << " ms, distance: " << m_nodes.Get(1)->
    GetObject<MobilityModel> ()->GetPosition().x << " m");
}

void UanExperiment::SendPacketTo0() {
    NodeContainer::Iterator node = m_nodes.Begin();
    Mac8Address dst = Mac8Address::ConvertFrom((*node)->
    GetDevice(0)->GetAddress());
    node++;

    string packet;
    ifstream in("/home/ttm4128/ccmcargopkt.txt", ios_base::in);
    in >> packet;
    int l = packet.length();
    in >> packet;
    l = packet.length();

    int i, j, k, b, p, janusIndex = -1;
    char newArr8[l / 8];
    uint8_t* janusDecArr = (uint8_t*) malloc(l / 8 * sizeof (
    uint8_t));
    j = -1;
    for (i = 0; i < l; i++) {
        j++;
        newArr8[j] = packet[i];
        if ((i + 1) % 8 == 0) {
            janusIndex++;
            j = -1;
            b = 0;
            for (k = 7; k > -1; k--) {
                p = (int) pow(2, 7 - k);
                b += chartobin(newArr8[k]) * p;
            }
            // NS_LOG_UNCOND("b = " << b);
            *(janusDecArr + janusIndex) = (uint8_t) b;
        }
    }

    in.close();

    Ptr<Packet> pkt = Create<Packet> (janusDecArr, l / 8);
    //cout << "timestamp at sendto0: " << timestamp << endl;
    Simulator::Schedule(Seconds(0), &UanExperiment::SendSinglePacket,
    this, *node, pkt, dst);
    timestamp += 0.0160203 + (double)l / 80;
    // Now + gen-enc + Janus encoding
    //NS_LOG_UNCOND("Sent packet to 0, timestamp: " <<
    to_string(timestamp));
}

void
UanExperiment::SendSinglePacket(Ptr<Node> node, Ptr<Packet> pkt,
Mac8Address dst) {
    //NS_LOG_UNCOND ( Simulator::Now ().GetHours () << "h" <<
    " packet sent to " << dst );
    PacketSocketAddress socketAddress;
    socketAddress.SetSingleDevice(node->GetDevice(0)->
    GetIfIndex());
    socketAddress.SetPhysicalAddress(dst);
    socketAddress.SetProtocol(0);
    m_sockets[node]->SendTo(pkt, 0, socketAddress);
}

void
UanExperiment::Prepare() {
    m_nodes.Create(2);
    SetupPositions();
    SetupCommunications();
    SetupApplications();
    SendPacketTo1();
}
```

```
int
main(int argc, char *argv[]) {
    CommandLine cmd(__FILE__);
    cmd.Parse(argc, argv);

    UanExperiment experiment;
    experiment.Prepare();

    Simulator::Stop(Days(6));
    Simulator::Run();
    Simulator::Destroy();

    experiment.Teardown();

    return 0;
}
```