NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

HARDENING AUTOMATIC IDENTIFICATION SYSTEMS: PROVIDING INTEGRITY THROUGH AN APPLICATION OF LIGHTWEIGHT CRYPTOGRAPHIC TECHNIQUES

by

Duc H. Nguyen

December 2020

Thesis Advisor: Chad A. Bollmann
Co-Advisor: Britta Hale

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE HARDENING AUTOMATIC IDENTIFICATION SYSTEMS: PROVIDING INTEGRITY THROUGH AN APPLICATION OF LIGHTWEIGHT CRYPTOGRAPHIC TECHNIQUES	**5. FUNDING NUMBERS**
6. AUTHOR(S) Duc H. Nguyen	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The Automatic Identification System (AIS) employed onboard maritime vessels was designed to improve the safety of navigation at sea, but focused on functionality rather than cybersecurity. Previous research has revealed that the AIS technical architecture and protocol have significant vulnerabilities that have the potential to facilitate cyber attacks such as spoofing and denial-of-service against AIS-equipped maritime vessels and port facilities. AIS data manipulation could have significant negative impacts on the global economy, regional geo-political stability, and safety-of-navigation at sea.

This thesis examines the technical and architectural feasibility of augmenting the current AIS architecture with data integrity and authentication capabilities to potentially mitigate AIS spoofing vulnerabilities. We assess the existing AIS architecture and lightweight cryptographic algorithms to determine an optimal, backwards-compatible authentication system. We then engineer a proof-of-concept AIS authenticator system using commercial AIS equipment and a physical testbench and demonstrate AIS message validation through public key digital signature verification.

14. SUBJECT TERMS Automatic Identification System, AIS, integrity, lightweight cryptography, spoofing, hijacking, availability disruption, ITU, NMEA, International Telecommunication Union, National Marine Electronics Association, public key infrastructure, PKI	**15. NUMBER OF PAGES** 231
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

HARDENING AUTOMATIC IDENTIFICATION SYSTEMS: PROVIDING INTEGRITY THROUGH AN APPLICATION OF LIGHTWEIGHT CRYPTOGRAPHIC TECHNIQUES

Duc H. Nguyen
Lieutenant, United States Navy
BS, U.S. Naval Academy, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN CYBER SYSTEMS AND OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
December 2020

Approved by: Chad A. Bollmann
Advisor

Britta Hale
Co-Advisor

Alex Bordetsky
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Automatic Identification System (AIS) employed onboard maritime vessels was designed to improve the safety of navigation at sea, but focused on functionality rather than cybersecurity. Previous research has revealed that the AIS technical architecture and protocol have significant vulnerabilities that have the potential to facilitate cyber attacks such as spoofing and denial-of-service against AIS-equipped maritime vessels and port facilities. AIS data manipulation could have significant negative impacts on the global economy, regional geo-political stability, and safety-of-navigation at sea.

This thesis examines the technical and architectural feasibility of augmenting the current AIS architecture with data integrity and authentication capabilities to potentially mitigate AIS spoofing vulnerabilities. We assess the existing AIS architecture and lightweight cryptographic algorithms to determine an optimal, backwards-compatible authentication system. We then engineer a proof-of-concept AIS authenticator system using commercial AIS equipment and a physical testbench and demonstrate AIS message validation through public key digital signature verification.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**

1.1 Congested Maritime Cyber-Physical Security Environment 1

1.2 Reliance on Cyber-Physical Systems and Challenges to Security in the Maritime Environment 2

1.3 Benefit to the Community 4

1.4 Overview 5

2 Background **7**

2.1 Automatic Identification System 7

2.2 Automatic Identification System Vulnerabilities 10

2.3 Standardization and Clarification of Technical AIS Nomenclature 13

3 Experimental Design and Considerations **17**

3.1 Research Questions 17

3.2 Security Design and Considerations 18

3.3 Proposed AIS Authenticator Mechanism and Design 20

3.4 Delivery Method 27

3.5 Choosing between Asymmetric or Symmetric Authentication Schemes . . . 37

3.6 Asymmetric Cryptographic Signature Algorithm Selection 42

3.7 Charm Cryptographic Library: Selection, Utilization, and Limitations . . . 48

4 A Necessary Alternate Research Methodology **51**

4.1 Implementation of an Identity-based Digital Signature(IBC_BLS04) 51

4.2 BLS04 Digital Signature Selection and Implementation 53

5 Experiment Design and Results **57**

5.1 Experiment Resign due to Equipment Design and Technical Challenges . . . 57

5.2 Testing Plan 62

5.3 Experimental Setup 67

5.4 Procedures, Results, and Phase Completion 73
5.5 System Metrics 82

6 Conclusions **97**
6.1 Research Questions 98
6.2 Recommendations and Areas for Future Work 99

Appendix A AIS Message Types and Description **105**

Appendix B Alphabet_Soup.PDF **109**
B.1 Overall AIS to NMEA Conversion Process 109
B.2 Detailed Breakout of the AIS to NMEA Conversion Process 109
B.3 Conversion of Digital Signatures to NMEA 6-bit ASCII 110
B.4 Detailed Breakout of DAC and FI Codes 110

Appendix C *KeyGenerationAuthority.PY* **115**

Appendix D *KeyGenerationAuthority.PY* Process Flow Graph **123**

Appendix E *InputEncoder.PY* **125**

Appendix F *InputEncoder.PY* Process Flow Graph **137**

Appendix G *A_Challenger.PY* **139**

Appendix H *A_Challenger.PY* Process Flow Graph **157**

Appendix I *B_Challenger.PY* **159**

Appendix J *B_Challenger.PY* Process Flow Graph **175**

Appendix K *A_Challenger.PY* CPU Load Data **177**

Appendix L ***B_Challenger.PY*** **CPU Load Data** **185**

Appendix M ***IBC_BLS04.PY*** **193**

Appendix N ***CertificatelessPKI.PY*** **197**

List of References **201**

Initial Distribution List **207**

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

Figure 1.1 The global trade routes that container ships utilize to transport trade goods around the world. Source: [5]. 2

Figure 1.2 Maritime voyage network interface on the bridge of *Oasis of the Seas*. Source: [9]. 3

Figure 2.1 A comparison of technical capabilities and limitations between AIS Class A and Class B. Source: [16]. 8

Figure 2.2 A Raymarine AIS950 Class A Transceiver (Only user-system interfacing unit shown). Source: [17]. 9

Figure 2.3 A Raymarine AIS350 Receiver. Source: [17]. 9

Figure 2.4 AIS Class A and Class B comparison of interoperability and limitations. Source: [19]. 11

Figure 2.5 An illustration of spoofed ship following a programmed path. Source: [2]. 12

Figure 2.6 A commercial shipboard network diagram. Source: [12]. 12

Figure 2.7 Visual representation of the differences between the adopted terminology (i.e., data packet, Message, and sentence). 14

Figure 3.1 Inter-protocol conversions and data exchanges in the current AIS architecture. 21

Figure 3.2 Inter-protocol conversions and communications in proposed design architecture. 22

Figure 3.3 Basic visual abstraction of a one-mechanism design architecture. 23

Figure 3.4 Basic visual abstraction of a two-mechanism design architecture. 24

Figure 3.5 Basic visual abstraction of a three-mechanism design architecture. 25

Figure 3.6	Abstraction of the proposed AIS authenticator's functions and mechanisms utilized to challenge and validate an AIS message between two ships.	27
Figure 3.7	Encapsulation and fragmentation for AIS data packets exceeding the 256-bit limit.	28
Figure 3.8	Detailed breakdown of AIS Message 6. Adapted from [20].	33
Figure 3.9	Detailed breakdown of AIS Message 8. Adapted from [20].	34
Figure 3.10	Detailed breakdown of AIS Message 25. Adapted from [20].	35
Figure 3.11	Detailed breakdown of AIS Message 26. Adapted from [20].	36
Figure 3.12	Goudossis's and Katsikas's proposed maritime identity-based cryptography architecture. Source: [21].	46
Figure 5.1	Basic outline of the layout and connections of all experiment tools and devices.	68
Figure 5.2	ALLTEK Marine: CAMINO-701 Class A AIS Transponder, Junction Box, and Laptop.	68
Figure 5.3	Junction Box and the "DISP" and "PILOT" serial connection ports.	69
Figure 5.4	FURUNO FA-150 connected via RS422 to Operator Laptop	69
Figure 5.5	Close up depiction of the RS422 Serial-to-USB converter.	70
Figure 5.6	The Faraday cage enclosure utilized to house the two RF antennas in order not to interfere nearby maritime traffic in Monterey Bay.	70
Figure 5.7	A closer look at the ports used to connect the two coaxial wires from the AIS transponders to the RF antennas inside the Faraday cage.	71
Figure 5.8	FA-150 serial COM ports. Only COM1 and COM2 were utilized.	71
Figure 5.9	FA-150 PC port.	72
Figure 5.10	FA-150 wiring diagram for COM1 and COM2 found underneath the removal cover. Utilized to connect to serial side of RS422 adapter.	72

Figure 5.11 FURUNO FA-150 technical manual depicting the supported NMEA Input and Output Messages specific to the available serial port connections. Source: FURUNO FA-150 Installation and Operation Manual. 74

Figure 5.12 A closer view of the CAMINO-701 junction box. 75

Figure 5.13 ALLTEK Marine CAMINO-701 technical manual depicting the supported NMEA Input and Output Message specific to the available serial port connections. Source: ALLTEK Marine CAMINO-701 Installation and Operation Manual edition 1.14. 75

Figure 5.14 CAMINO-701 (laptop) display of received AIS data packets from FA-150 transponder. 78

Figure 5.15 FA-150 (laptop) display of received AIS data packets from CAMINO-701 transponder. 79

Figure 5.16 A display of *InputEncoder.py*'s process readouts regarding the data inputs, conversions, and outputs. 80

Figure 5.17 A partial display of Outfile.txt. This shows a sample of the data strings converted from simple 8-bit ASCII to final AIS VHF Datalink Message (AIVDM) sentences that are expected on the CAMINO-701 receiving end. 81

Figure 5.18 Various process states of Phase 2. 83

Figure 5.19 A display of B_Challenger's readout indicating the receipt of a challenge message from the CAMINO-701 transponder followed by the digital signature creation process. 84

Figure 5.20 This figure depicts the five digital signature fragments originating from the FA-150. 85

Figure 5.21 This figure depicts the successful reassembly and verification of the five digital signature fragments. 86

Figure 5.22 AIS authenticator test data from 60 iterations. 88

Figure 5.23 Class A AIS shipboard mobile equipment reporting intervals based on various dynamic vessel states. Source: [7]. 89

Figure 5.24 Class B AIS shipboard mobile equipment reporting intervals based on various dynamic vessel states. Source: [7]. 90

Figure 5.25 Recorded CPU load of the Virtual Machine (VM) assigned to the CAMINO-701. CPU load was recorded under three conditions: passive, idle, and active. 93

Figure 5.26 Recorded CPU load of the VM assigned to the FA-150. CPU load was recorded under three conditions: passive, idle, and active. . . 94

Figure 5.27 Key statistics for recorded CPU load percentages of the VM assigned to the CAMINO-701. CPU load was recorded under three conditions: passive, idle, and active. 95

Figure 5.28 Key statistics for recorded CPU load percentages of the VM assigned to the FA-150. CPU load was recorded under three conditions: passive, idle, and active. 95

Figure D.1 *KeyGenerationAuthority.py* Process Flow Graph 124

Figure F.1 *InputEncoder.py* Process Flow Graph 137

Figure H.1 *A_Challenger.py* Process Flow Graph 157

Figure J.1 *B_Challenger.py* Process Flow Graph 175

List of Tables

Table 3.1 A table of various considerations that affect the proposed AIS authenticator's security design, implementation, and authentication capabilities. 18

Table 3.2 Maximum AIS payloads for specific AIS Message types. Adapted from [7]. 29

Table 3.3 A table detailing an application identifier bit 0's through 15's fit and function. Adapted from [7]. 30

Table 3.4 Tabular summary of 64 AIS Application Specific Messages (ASM). Adapted from [24]. 31

Table 4.1 Various maximum sizes of identity-based cryptographic digital signature based on the IBC_BLS04 scheme [40]. 52

Table 4.2 Digital signature length comparison utilizing various available digital signature schemes and pairing groups from Charm Cryptography Library. Note that the "blacked-out" portions represent the combinations that resulted in a system fragmentation fault and did not produce a signature. 55

Table 5.1 Summary of averaged recorded metrics for proposed system performance over 60 transmissions. 87

Table 5.2 Summary of overall system performance over six trial sets (60 transmissions). 88

Table A.1 AIS message types and their descriptions. Adapted from: [20]. . . 105

Table K.1 *A_Challenger.py*'s Measured CPU Performance Over Time 177

Table L.1 *B_Challenger.py*'s Measured CPU Performance Over Time 185

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

ABM Addressed Binary Messages

AIS Automatic Identification System

AIVDM AIS VHF Data-link Message

ASM Application Specific Messages

BBM Broadcast Binary Messages

CIANA Confidentiality, Integrity, Availability, Non-repudiation, Authenticity

COTS Commercially-off-the-shelf

CPN Chart Plotting Navigation

CPS Cyber Physical System

CRL Certificate Revocation List

DAC Designated Area Code

DISA Defense Information Systems Agency

DoD United States Department of Defense

DSA Digital Signature Algorithm

ECDSA Elliptical Curve Digital Signature Algorithm

EEMS Engineering Enclave for Maritime Security

EMS Electromagnetic Spectrum

FI Function Identifier

FIPS Federal Information Processing Standards

GMSK Gaussian-filtered minimum shift keying

GPS Global Positioning Satellite System

IALA International Association of Marine Aids to Navigation and Lighthouse Authorities

IMO International Maritime Organization

IP Internet Protocol

ITDMA Incremental Time Division Multiple Access

ITU International Telecommunications Union

KGC Key Generation Center

MAC Message Authentication Code

MMSI Maritime Mobile Service Identity

NIST National Institute of Standards and Technology

NMEA National Marine Electronics Association

NPS Naval Postgraduate School

NRZI Non-return-zero-inverted

OCO offensive cyber operations

OCSP Online Certificate Status Protocol

PKI Public Key Infrastructure

RATDMA Random Access Time Division Multiple Access

RF Radio-frequency

SAR Search and Rescue

SDR Software-Defined Radios

SOLAS International Convention for the Safety of Life at Sea

SOTDMA Self Organized Time Division Multiple Access

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol / Internet Protocol

TDMA Time-Division Multiple Access

UDP User Datagram Protocol

VDE VHF Data Exchange

VDM VHF Data Message

VHF Very High Frequency

VM Virtual Machine

WRC World Radiocommunication Conference

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

First, I wish to express my deepest gratitude to my thesis advisors. Without your help, guidance, technical expertise, patience, professionalism, dedication, and push I would definitely would not have finished this research or thesis. Professors, you forced me outside of my comfort zone and forced me to look beyond my doubts and frustrations. Thank you for challenging, guiding, and believing in me.

I wish to acknowledge the great support of the Naval Postgraduate School Graduate Writing Center. Michelle and John, thank you very much for all your help, patience, support, and tutelage. You taught a stubborn engineer that there is more to English writing than bullet points and lists. My academic writing improved significantly. Thank you!

I would also like to give special regard the Thesis Processing Office. Sandi and Rebecca, thank you for the sincere support during the final weeks of my time at NPS. You helped me keep my head up and feet moving.

Finally, I must express my very profound gratitude and love to my dear friends for providing me with unfailing emotional support and continuous encouragement throughout my time at NPS. Mark, Silva, Zeke, Ira, Ryan, Rachel, Connie, and Cha Siu Bao: you made my days brighter and full of happiness and laughter. Thank you.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

A significant vulnerability exists in the Automatic Identification System (AIS). The AIS is an internationally used maritime radio-broadcast system designed to improve safety of navigation at sea and does not incorporate any form of information assurance or cyber-physical security.

This thesis proposes an AIS-specific application of a lightweight cryptographic algorithm to potentially decrease the vulnerable attack surface area and thereby improve safety of navigation at sea. The primary objective and goal of this research is to provide a proof-of-concept for an AIS message authentication system that adds message integrity and authentication capabilities. The need for this capability is illustrated by the recent AIS spoofing attack in Shanghai [1] as well as other attack vectors identified in [2].

This research attempts to answer the following three questions: What should the overall AIS technical architecture look like after the augmentation of data integrity and authentication capabilities? Does the current AIS protocol possess sufficient flexibility to provide data integrity and authentication without changing the overall technical architecture? To what extent does the current AIS protocol and standard constrain the list of suitable cryptographic algorithms?

1.1 Congested Maritime Cyber-Physical Security Environment

Worldwide, there are approximately 400 maritime liner shipping services in operation [3]. These vessels transport 60 percent of all goods, by value, each year across the vast expanse of ocean and territorial waters to feed the global economy [3]. In 2014, the top three producers (China, the United States, and the Republic of South Korea) exported nearly 54 million shipping containers combined, a statistic accounting for approximately 42 percent of all global containerized shipping [4].

Contrary to public perception, global trade routes occupy a minor fraction of the total

oceanic surface area due to the limited number of cost-efficient maritime trade routes, as shown in Figure 1.1. As a result of these common trade routes and the resultant high-density traffic areas (e.g., Dover, Hormuz, Malacca, Gibraltar), an opportunity exists for malicious attackers to inflict significant damage to multiple maritime vessels through a single AIS-directed cyber attack; [2] discusses various possible examples of AIS-directed cyber attacks.

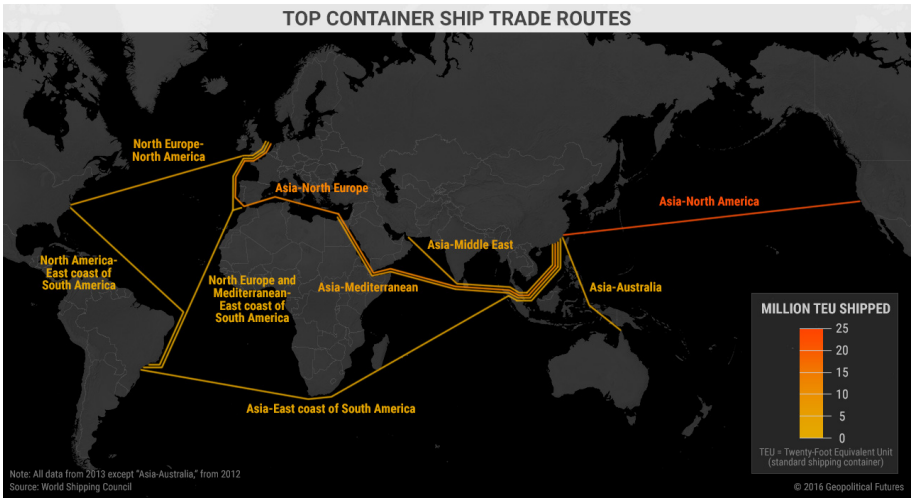


Figure 1.1. The global trade routes that container ships utilize to transport trade goods around the world. Source: [5].

1.2 Reliance on Cyber-Physical Systems and Challenges to Security in the Maritime Environment

Since 2000, the International Maritime Organization (IMO) required, through the International Convention for the Safety of Life at Sea (SOLAS), certain vessels "to carry AIS capable of providing information about the ship to other ships and coastal authorities automatically" [6]. This international mandate facilitated the critical exposure necessary for malicious actors to exploit any one of several possible attack vectors.

Maritime vessels and their crews rely heavily upon sophisticated Cyber Physical System (CPS) networks of specialized computers, application suites, and ship systems to safely operate and navigate the narrow trade corridors and cross the oceans. However, the combination of cybersecurity deficiencies, undocumented variations, and heavy international use

introduce risks to the AIS. AIS-specific risk to maritime trade is the composite product of 1) vulnerabilities associated with the current AIS standards and manufacturer practices and 2) threats derived from exposure to capable malicious actors due to the system's mandate for international use.

Figure 1.2 depicts an example of the array of maritime cyber-physical interface stations aboard the cruise ship *Oasis of the Seas*. The systems that comprise these complex CPS networks do not incorporate any form of cybersecurity protections (i.e., confidentiality, integrity, authenticity, non-repudiation, or availability) and often vary by design and capabilities depending on both the manufacturer and the installation location.

Manufacturers must conform to the International Telecommunications Union (ITU)'s ITU-R M.1371-5 [7] and National Marine Electronics Association (NMEA)'s NMEA-0183 [8] published standards. The ITU-R M.1371-5 pertains mostly to the AIS protocol's use and propagation through the electromagnetic spectrum. Conversely, the NMEA-0183 provides manufacturers with technical specifications for maritime voyage network intercommunications, design, and implementation.

Figure 1.2. Maritime voyage network interface on the bridge of *Oasis of the Seas*. Source: [9].

Although the AIS and NMEA standards provide the technical and operational parameters

for shipboard equipment, the standard is only guidance and a set of recommendations. Manufacturers are not bound to the standard and are free to design and engineer maritime systems using proprietary technology as they see fit.

Each variation from the standard has the potential to create a new attack vector or vulnerability, especially when the variation goes undocumented for various reasons. Consequently, the AIS community of interest is unable to properly address and correct these possible system vulnerabilities if the community is ignorant of the undocumented deviations. Another issue exists when third-party developers and researchers have specific technical expectations, derived from published standards, but encounter operational and technical abnormalities due to undocumented, vendor-specific deviation from the standard, as discussed in Chapter 5.

1.3 Benefit to the Community

This research attempts to contribute to the maritime community by adding both data integrity and authentication, two of the five pillars of information assurance (i.e., confidentiality, integrity, availability, non-repudiation, and authenticity), to the current AIS technical architecture. Additionally, the Department of the Navy would benefit from this research in similar regards with the maritime community. The Department of the Navy would gain an AIS message validation capability (i.e., the ability to verify the validity of the data source and integrity). The National Institute of Standards and Technology defines integrity as the "property whereby data has not been altered in an unauthorized manner since it was created, transmitted or stored" [10] and authenticity as the verification of "the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system" [11].

Our proposed contribution supports novel AIS capabilities that would permit properly-equipped maritime vessels to discriminate against malicious AIS traffic. These data integrity and authentication capabilities would potentially reduce the attack surface area associated with AIS vulnerabilities, which facilitates attacks such as spoofing, denial-of-service, hijacking, and availability disruptions.

1.4 Overview

This research explored the feasibility of augmenting the current AIS technical architecture with data integrity and authentication capabilities through an externally-integrated, modular, and Commercially-off-the-shelf (COTS)-sourced AIS authenticator system consisting only of a laptop computer and associated software. Specifically, this research explored the combination of lightweight cryptographic algorithm options, secret key distribution system options, and the necessary system architecture that would provide an application-specific hardware and software solution to potentially mitigate known AIS and NMEA vulnerabilities.

This research designed, implemented, evaluated, and provided a proof-of-concept for an AIS data validation system which incorporates an existing lightweight cryptographic algorithm compatible with NMEA and ITU standards and recommendations.

The remainder of this thesis is organized as follows: Chapter 2 describes necessary information regarding AIS background, device types, vulnerabilities, and prior related work, which focused on specific NMEA and ITU vulnerabilities and possible attack vectors. Chapter 3 is a discussion of the proposed experimental design and considerations to include possible lightweight cryptographic schemes and frameworks, message delivery methodology, technical and design constraints, and related research. Chapter 4 discusses the unique challenges and limitations of available lightweight cryptographic algorithms, architectural constraints, cryptographic tool sets and resources, and our subsequent compromises and alternate design proposition. Chapter 5 discusses the experimental methodology, results, unexpected discoveries and abnormalities, and system performance. Lastly, Chapter 6 concludes our research with discussions of our recommendations, possible future work, and experimental weaknesses.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

This chapter provides background information on AIS use, carriage requirements, and system types. Additionally, this chapter details previous research on the syntactical and architectural vulnerabilities of the AIS and NMEA protocols.

Previous research (e.g., [2], [12]–[14]) established that the overall AIS architecture is vulnerable to external threats. These four works identified and concluded that the current AIS protocol does not feature any information assurance mechanisms or capabilities to discriminate and protect against malicious data. Additionally, AIS data transmissions, regardless of function, are accepted and processed if formatted according to ITU and NMEA syntax structures [2], [12], [13]. Therefore, malicious actors may take advantage of this vulnerability to conduct user-defined data injects by disguising malicious data as properly formatted AIS messages [2], [13].

2.1 Automatic Identification System

AIS is an automated suite of transceivers and specialized application systems designed for use in the maritime environment. AIS is intended to improve maritime safety and navigation by facilitating the efficient exchange and broadcast of maritime static and dynamic navigational data between ships, Search and Rescue (SAR) aircraft, and shore stations [15]. In 2000, SOLAS Regulation 19 mandated the use of AIS technology on specific maritime vessels:

> All ships of 300 gross tonnage and upwards engaged on international voyages and cargo ships of 500 gross tonnage and upwards not engaged on international voyages and passenger ships irrespective of size shall be fitted with an automatic identification system [6].

There are three basic types of AIS categories: A, B, and receive-only. Depending on the category, the subject AIS system will have varying capabilities and technical requirements, as illustrated in Figure 2.1.

Shipborne AIS Comparison	Class A	Class B-SO	Class B-CS
Primary Access Scheme	Self-Organizing Time-Division Multiple Access (SO-TDMA per ITU-R M.1371)		Carrier-Sense TDMA (CS-TDMA)
Frequency Range	156.025–162.025 MHz (25 kHz bandwidth)		161.500–162.025 MHz (25 kHz bandwidth)
Digital Selective Calling	Dedicated receiver	Time-shared with a TDMA receiver	
Transmit Power	12.5 Watts (1 W low-power)	5 Watts (2 W low-power)	2 Watts only
Positioning Source	Interfaced to vessel's primary Electronic Positioning Fixing System; Internal Global Navigation Satellite System (GNSS) as a fall-back	Internal GNSS	
Position Reporting	Every 2 s if >23 kts; 3.33 s if >5° course change; 6 s if >14-23 kts; 10 s if 2-14 kts; 3 min. if at anchored, moored, or =< 3 kts; via Message 1	Every 5 s if >23 kts; 15; if 14-23 kts; 30 s if >23 kts; 3 min. if at anchored, moored, or =< 2 kts; via Message 18	Via message 18; every 30 s (±4 s), subject to slot availability; 3 min. if at anchored, moored, or =< 2 kts; via Message 18
	Message 1 reports: MMSI, Time-stamp, Position, Position Accuracy flag, RAIM flag, COG, SOG, HDG, ROT, Navigation Status, Communication State; Message 18 omits ROT and Navigation Status, but, adds various Class B flags for: Type (SO/CS), Operating Mode, and, availability of a Display, DSC Receiver, Full/Limited Bandwidth, and Channel Management		
Static & Voyage Data Reporting	Every 6 min. via Message 5	Every 6 min. via Message 24A&B	
	Message 5 reports: MMSI, IMO#, Call-sign, Name, Ship Type, Dimensions, Static Draft, Destination, ETA, EPFS type, Data Terminal availability, AIS version; Message 25A&B omits IMO#, Static Draft, Destination, ETA, Data Terminal availability, AIS version but, adds Vendor ID.		
Application & Safety Text Messaging	Receive & transmit	Receive optional, cannot transmit	
Display & Interfacing	Minimal Keyboard Display (MKD) required Two input-output ports; multiple interfaces	Display optional; One input-output interface	Both optional
Test Standard	IEC 61993-2	IEC 62287-2	IEC 62287-1
USCG Approval Nr.	USCG 165.155/x/x	USCG 165.157/x/x	USCG 165.156/x/x
Estimated Cost	$2,600-4,000	$2,000	$700-1,600

Figure 2.1. A comparison of technical capabilities and limitations between AIS Class A and Class B. Source: [16].

Figure 2.1 vertically lists the AIS specifications on the left margin and the AIS class types across the top margin. For clarification, Class B-SO is Class B - Self Organizing Time-Division Multiple Access (TDMA) and Class B-CS is Class B - Carrier Sense. The differences between these two subcategories are not relevant to this research. The figure was enlarged and rotated to enhance readability.

Class A AIS systems are the most capable, complicated, and costly (an example is shown in Figure 2.2); conversely, Class B systems are a lower-cost and less capable alternative for vessels not requiring a Class A system [17], [18]. Receive-only AIS devices, an example shown in Figure 2.3, are primarily for recreational vessels whose operators require situational awareness for safety of navigation but do not necessarily need to transmit their voyage data. Figure 2.4 lists the inter-communication capabilities and limitations of each AIS class.

Figure 2.2. A Raymarine AIS950 Class A Transceiver (Only user-system interfacing unit shown). Source: [17].

Figure 2.3. A Raymarine AIS350 Receiver. Source: [17].

2.2 Automatic Identification System Vulnerabilities

Previous research has shown that AIS is vulnerable to spoofing, hijacking, and availability disruption [2]. These attacks are possible because the current AIS technical architecture and standard does not incorporate the Confidentiality, Integrity, Availability, Non-repudiation, Authenticity (CIANA) pillars of information assurance; consequently, any receiving vessel will trust and accept a properly-formatted AIS message regardless of its payload or originator [2], [12], [13]. A malicious actor could utilize this security flaw to negatively affect maritime safety and navigation, global commerce, and international politics, such as in the Shanghai incident [1].

2.2.1 Message Syntax and Architecture Vulnerabilities

Trend Micro, a multinational cyber security company with global headquarters in Tokyo, Japan, reported in 2014 that AIS is vulnerable to spoofing, hijacking, and availability disruption [2]. Their experiments confirmed and demonstrated the ease of AIS data packet manipulation, replay attacks, and obfuscation via an Software-Defined Radios (SDR) and an open source software. Trend Micro concluded that malicious actors could cause significant economic and political damage globally, given the current AIS standard and technical architecture.

In Trend Micro's research, spoofed AIS data packets that encapsulated a fictitious ship's navigational track-over-ground, were broadcasted to a test receiver. The test receiver accepted the spoofed AIS data packets and plotted the fictitious ship's course. This stimulated attack, when scaled to the real world, could potentially escalate political tensions between rival governments when tracks of fictitious commercial or military ships venture into restricted territorial waters or exclusive economic zones. Figure 2.5 represents the track-over-ground of Trend Micro's fictitious ship in the Mediterranean Sea.

Previous work at the Naval Postgraduate School (NPS) explored AIS to research the feasibility of maritime voyage network exploitation through SDR and open-source software utilization. Rubinstein [12] identified that "modern maritime vessels are floating [cyber-physical systems]" and categorized the typical commercial voyage network using three main functional areas: 1) voyage network, 2) engineering network, and 3) administrative network, as shown in Figure 2.6. These voyage networks utilize complex embedded networks

to monitor, regulate, and actuate critical shipboard functions.

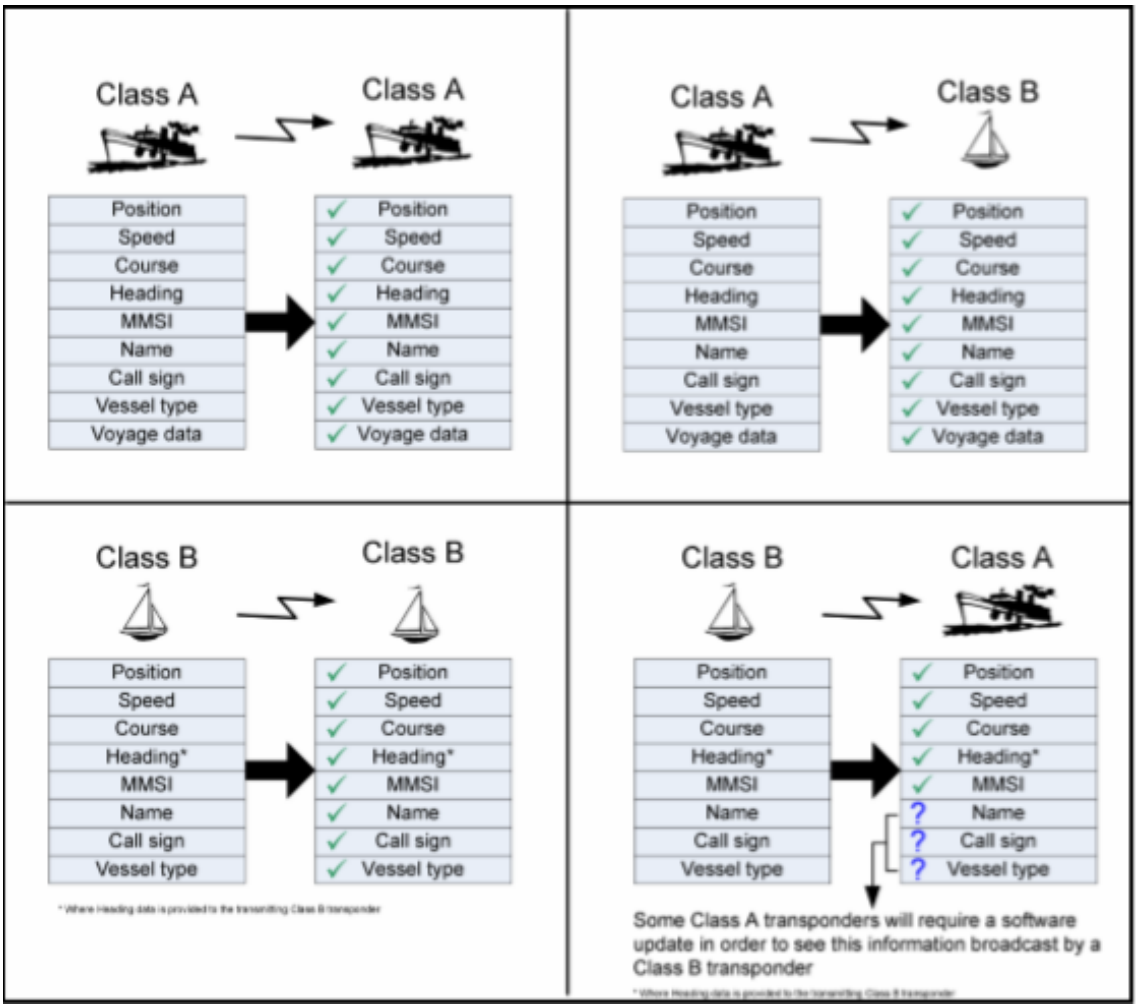


Figure 2.4. AIS Class A and Class B comparison of interoperability and limitations. Source: [19].

Rubinstein specifically explored how the NMEA-0183 standard affects maritime voyage networks and causes AIS to be vulnerable. He concluded that shipboard voyage networks are critically vulnerable to direct AIS message injections via radio-frequency apertures [12]. These integrated networks are capable of interfacing with each other but often via proprietary communication methods. Consequently, manufacturer design freedoms and NMEA architectural similarities between individual ships introduced "unintended access through common commercial systems" [12].

Figure 2.5. An illustration of spoofed ship following a programmed path. Source: [2].

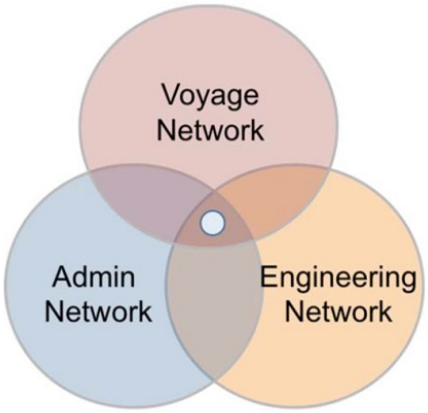


Figure 2.6. A commercial shipboard network diagram. Source: [12].

Using an offensive cyber operations (OCO) approach, Rubinstein [12] developed an effective experimental "AIS security research tool" to reverse engineer and replicate named NMEA Messages. He successfully spoofed an experimental AIS network by injecting NMEA-formatted, arbitrary, user-defined data strings.

His methodology included a custom Python script used to inject arbitrary, user-defined static and dynamic navigational data into the experimental NMEA data streams. The arbitrary data strings were formatted per ITU-R M.1371-5 and NMEA-0183 standards and were validated for system acceptability through the transmitting AIS transponder. The test

receiver accepted, parsed, and plotted the arbitrary data without error onto the Chart Plotting Navigation (CPN) computer, producing results analogous to those in Figure 2.5.

In 2017 Heaney [13] extended Rubinstein's research by exploring the "weaknesses in the AIS protocol that allow for targeted or broadcast distribution of binary data to any AIS". He applied Rubinstein's research to the concept of AIS "network amalgamation" of disparate maritime equipment, which all utilize the NMEA-0183 standard to communicate both internally and externally to the ship. Specifically, Heaney explored the issue of "unintended access through common commercial systems" [12] through arbitrary, user-data injections. He concluded that it is theoretically possible to gain unauthorized access to any networked NMEA-0183 compliant system utilizing malicious, user-data injections made possible through AIS-specific binary data Messages (i.e., Message 6, 8, 25, and 26). His theory is analogous to a potential cyber-attack scenario whereby a malicious payload establishes a reverse Transmission Control Protocol (TCP) shell with inherited privilege and access to the attacker. In a maritime context, one possible scenario is when an attacker establishes a remote access connection to any critical navigation system or subsystem to execute code via AIS binary data Messages [13].

In 2019, Harris reported that AIS spoofing is a common occurrence around Shanghai and that many vessels conduct these activities to avoid interdiction by local maritime authorities while transporting illicit goods (e.g., sand dredged from the Yangtze River) [1]. Harris highlighted an event when the captain of MV *Manukai* reported strange AIS data abnormalities while transiting through the Chinese territorial waters surrounding Shanghai [1]. While docked, the MV *Manukai* discovered that its identity and public maritime information were propagating from a spoofed nearby location. Similarly, other instances of spoofed AIS data involved ship voyage information suggesting transit through the Huangpu River while the physical ship was located on the Yangtze River a few miles away [1].

2.3 Standardization and Clarification of Technical AIS Nomenclature

This section provides standard nomenclature for the use of "message", "sentence", and "packet" for the succeeding chapters of this thesis.

The ITU-R M.1371-5 and NMEA-0183 utilize the terms “message” and “sentence” interchangeably. Often, “message” and “sentence” are used to refer to named AIS Message types; e.g., Message 1, Message 2; NMEA input/output sentences, e.g., ABM, ACK, BBM, DTM; AIS and NMEA encapsulated data; fully formatted AIS and NMEA data transmissions; sentence/message fragmentation sequencing; cumulative sentence counters; and multi-message sentence sequencing. Effectively, there is no proper uniformity or standardization of terminology between the two standards. If a specific section of either standards’ documentation is not explicit with its terminology, contextual clues are typically required for clarification.

For clarity throughout this work, we adopt the following terminology and provide Figure 2.7 as a visual aid regarding our chosen terminology. This figure details a scenario that features an AIS Message 6 requiring fragmentation and transmission via four AIS data packets.

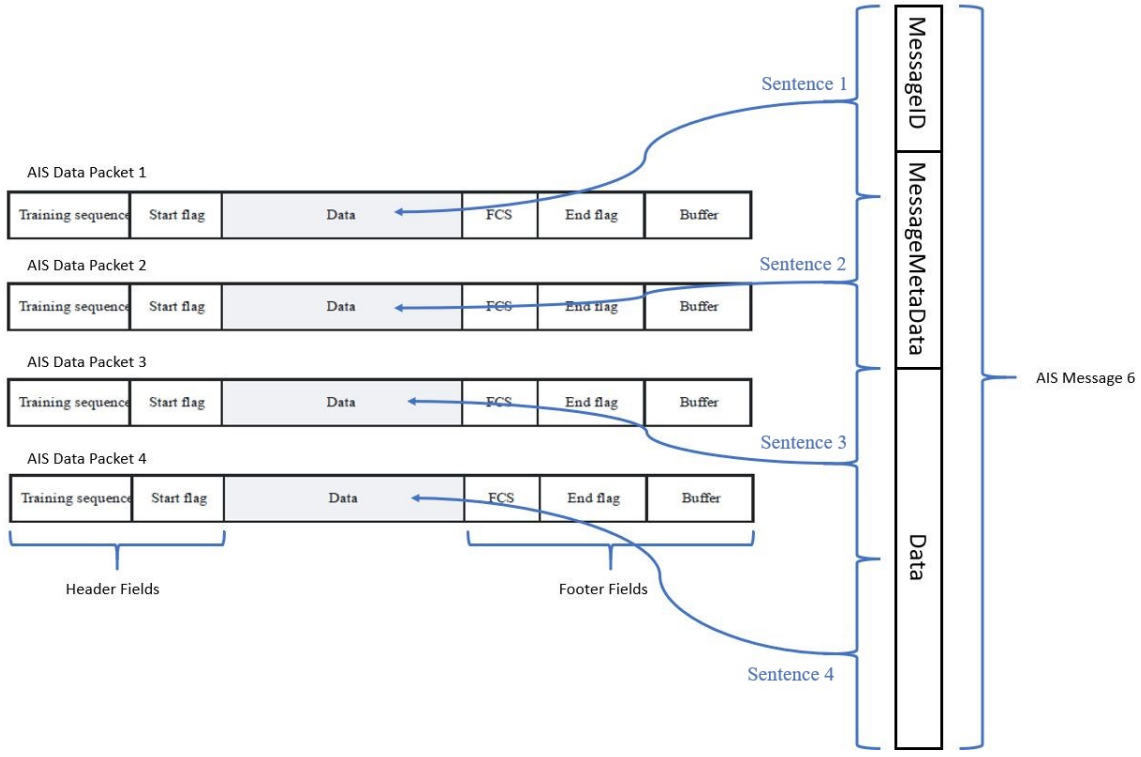


Figure 2.7. Visual representation of the differences between the adopted terminology (i.e., data packet, Message, and sentence).

“Message”, a proper noun, refers to all specifically named and implemented AIS Message

types, e.g., Message 1, Message 2, Message 3, and NMEA-0183 Input/Output Messages, e.g., ABM, ACK, BBM, DTM.

"Data packet(s)" or "packet(s)" refer(s) to a properly-formatted AIS transmission. A packet is the transmission-ready data that includes all header, footer, and encapsulated AIS and NMEA information. Example: "The current AIS or NMEA protocols and standards do not offer data integrity or authentication capabilities to transmitted or received AIS data packets."

"Sentence(s)" refer(s) to the encapsulated data located between the AIS header and footer fields inside an AIS data packet. Sentences are strictly used in situations involving the NMEA data to be encapsulated exceeds the payload size limitation of an AIS data packet; the individual NMEA data fragments are referred as sentences. Multi-sentence AIS Messages are typically of types 6, 8, 12, and 14. Example: "AIS Message 6 is one of many user-definable AIS Message types. Often these Message 6 exceeds the payload size limitation of an AIS data packet and requires fragmentation under NMEA and ITU rule sets for handling multi-sentence AIS messages."

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Experimental Design and Considerations

The central hypothesis of this thesis is that the current AIS protocol and standard possess sufficient flexibility to add a novel data integrity and authentication capability through an external and modular AIS authentication system. This experimental AIS authentication system should incorporate specific design features and mechanisms which align with a number of AIS technical and operational constraints. These constraints include system scalability, compliance with standards, and bandwidth constraints while underway.

This chapter details the research questions, technical considerations, design methodology, and application of cryptography required for the creation of a functional AIS message authentication system which utilizes AIS Message 6 and Message 8 to encapsulate and deliver authenticated AIS data packets.

3.1 Research Questions

This thesis intends to answer the following three research questions.

What would the overall AIS technical architecture look like after the addition of data integrity and authentication? Integrity and authenticity are novel and inorganic AIS capabilities; current commercially-available AIS devices do not support these two pillars of information assurance.

Does the current AIS protocol possess sufficient flexibility to provide data integrity and authentication without changing the overall AIS technical architecture? The ITU-R M.1371-5 and NMEA-0183 standards provide the technical and operational recommendations for the AIS. This research was restricted to the existing capabilities and protocols documented in these two standards. This research does not desire to introduce foreign technologies to the overall design and risk standard nonconformity and non-functionality.

To what extent do the ITU-R M.1371-5 and NMEA-0183 standards constrain the population of suitable cryptographic algorithms? This research explored and implemented

lightweight cryptographic algorithms to determine the extent of the AIS's compatibility and feasibility of supporting cryptography.

3.2 Security Design and Considerations

This research identified various considerations that directly influence the proposed AIS authenticator's security design, implementation, and authentication capability. These considerations are discussed in Table 3.1.

Table 3.1. A table of various considerations that affect the proposed AIS authenticator's security design, implementation, and authentication capabilities.

Proposed Authenticator Design Constraints and Considerations	
Constraint	Information
Scalability	The proposed system must operate aboard every AIS-equipped vessel concurrently, scale proportionally with the growth or decline of the maritime community, and support operations in high-density traffic environments.
Low Computational Cost	Queried AIS transmissions increase the computational overhead associated with the proposed solution. The proposed design architecture must utilize cryptographic schemes and algorithms that are optimized and efficient to support operation in high-density maritime traffic areas such as the straits of Malacca, Dover, and Gibraltar.

Table 3.1 continued from previous page

Confidentiality	The proposed design architecture will not provide confidentiality to AIS message traffic. AIS's primary function is to provide safety of navigation through direct or broadcast maritime static and dynamic voyage data. Confidentiality would detract from the AIS's intended purpose if other vessels are unable to decrypt the ciphertext. Furthermore, confidentiality would potentially interfere with maritime assistance if the assisting vessel is not properly equipped to decrypt the AIS ciphertext.
Forward Compatibility	The proposed design architecture must possess a characteristic that allows for transfer to and integration with all future AIS policy, hardware, or capability upgrades.
Compliance with NMEA-0183 (series) standard	The system design must conform and operate within the limits of this standard. Specifically, this standard applies to how voyage information and data structures are utilized and shared across multiple embedded maritime devices.
Compliance with ITU-R M. 1371 (series) standard	The ITU-R M.1371-5 standard provides the guidance necessary for Radio-frequency (RF) communication between AIS stations. This is a design constraint that directly influences which cryptographic algorithm would be possible for design implementation due to AIS packet size limitations.
Modular Design	The overall design architecture must be highly modular to provide a "plug and play" capability. Its removal must not have any negative consequences for the embedded shipboard network.
Bandwidth Economy	The ITU-R M.1371-5 standard imposes bandwidth requirements on how much data any AIS station may transmit. The proposed system design must account for, comply with, and adapt to the 27 existing AIS Messages [20]. Appendix A details the 27 different AIS Messages and their use.

Table 3.1 continued from previous page

Minimalist System	The proposed system must only utilize protocols, processes, and capabilities organic to the current AIS and NMEA standards. Manufacturer adoption is more likely with minimal invasiveness to the current AIS architecture.
Receive-only AIS Users	This design consideration prioritizes both Class A and Class B AIS devices over receive-only AIS devices. Research efforts will focus primarily on the requirements and capabilities associated with both Class A and Class B AIS transceivers.
Bandwidth Constrained	Typically, vessels in port have higher bandwidth availability than underway vessels. RF propagation is the only means available to an underway vessel for off-ship communications. The proposed system should employ a message validation mechanism (i.e., key management and acquisition) that does not burden the vessel's bandwidth limitations.

With regards to confidentiality, some authors [21] argue for the benefits of confidentiality in areas where pirates have been known to operate. Although the argument is sound, this research argues that current mitigation and protection mechanisms provide layers of defense-in-depth against possible pirate attacks (e.g., route diversion, on-board security personnel, and Combined Task Force 151 anti-piracy efforts [22]).

This research acknowledges and agrees with Bruce Scheiner's assessment and *caveat emptor* regarding the advancement of technology and its impact on the notion of cryptographic bit-size security: Cryptographic algorithms or mechanisms used today could be obsolete tomorrow [23]. Thus, forward compatibility is a necessity for the longevity of this proposed system design.

3.3 Proposed AIS Authenticator Mechanism and Design

The overall proposed AIS authenticator design integrates a modular security system into the current AIS architecture. This novel security system has the potential to mitigate

against some possible attacks (discussed in [1], [2]) by means of cryptographic protection of outgoing AIS data packets.

Figure 3.1 is a high-level, component-based, visual abstraction of how a common, integrated shipboard voyage network currently communicates to another using the current AIS architecture. The middle blocks (i.e. binary, NMEA-0183, and ITU-R M.1371-5) represent the data state required by each shipboard component (i.e., Shipboard System, Shipboard Network, and AIS device).

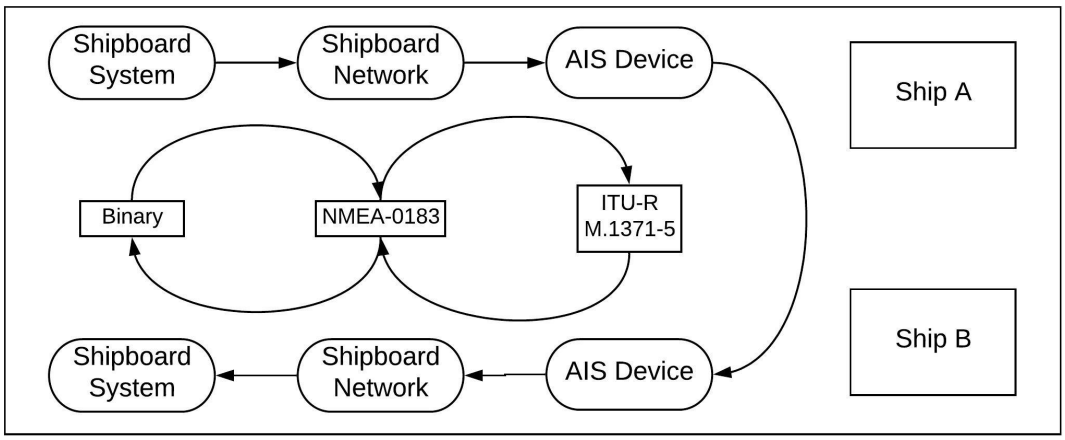


Figure 3.1. Inter-protocol conversions and data exchanges in the current AIS architecture.

The proposed AIS authenticator requires an architecture that resembles and incorporates capabilities depicted in Figure 3.2. The difference between Figure 3.1 and Figure 3.2 is the incorporation of an "authenticator" block between the "shipboard network" and "AIS device" blocks.

The "authenticator block" is analogous to a transponder, adding supplemental responses and capabilities to the AIS that either permit or block received AIS data packets based on user-implemented rule sets.

During experiments in this work, the proposed authenticator was instantiated via a standalone laptop and several Python scripts. The proposed AIS authenticator functions as a broker and virtual "demilitarized zone." It receives and passes data from the ship's voyage

network to the AIS transponder and vice versa. The AIS authenticator filters and protects the integrated voyage network from illegitimate data by only allowing validated data packets to reach the voyage network from the AIS transponder.

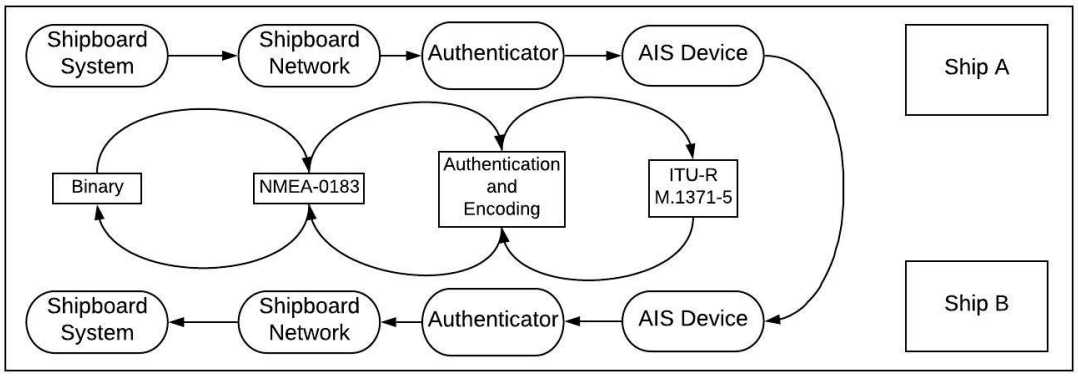


Figure 3.2. Inter-protocol conversions and communications in proposed design architecture.

There are multiple conceivable approaches to accomplishing authentication. The trade-offs associated with a particular AIS authenticator design can be evaluated by considering the number of mechanisms required to accomplish authentication.

3.3.1 One-Mechanism Design Architecture

Figure 3.3 depicts a high-level, visual abstraction of a one-mechanism AIS authenticator design. A one-mechanism architecture would require the combined encapsulation of the normal AIS data packet and its cryptographic value into the originally-intended AIS Message type. For example, if Vessel A intends to transmit AIS Message 1 (i.e., an AIS Position Report broadcast), the AIS transponder would have to encapsulate the navigational voyage data and its cryptographic value within the technical parameters of AIS Message 1.

A one-mechanism design is not an optimal choice because this design violates the ITU-R M.1371-5. Each AIS Message type has a specific task, purpose, and payload data specifications and size limitation. The incorporation of a cryptographic value with the original AIS message's payload would require modifying the ITU-R M.1371-5's regulation of the AIS message's syntax and composition. AIS Message 1 is for navigational data

reports; its data payload is a constant 168 bits and the overall AIS data packet size is 256 bits. These values are constant regardless of the maritime environment.

Furthermore, a one-mechanism design would not be feasible with other AIS message types that feature a variable message payload because the one-mechanism solution would need to function with all 27 AIS Message types. Additionally, the majority of the 26 other AIS Message types do not permit variable message payloads.

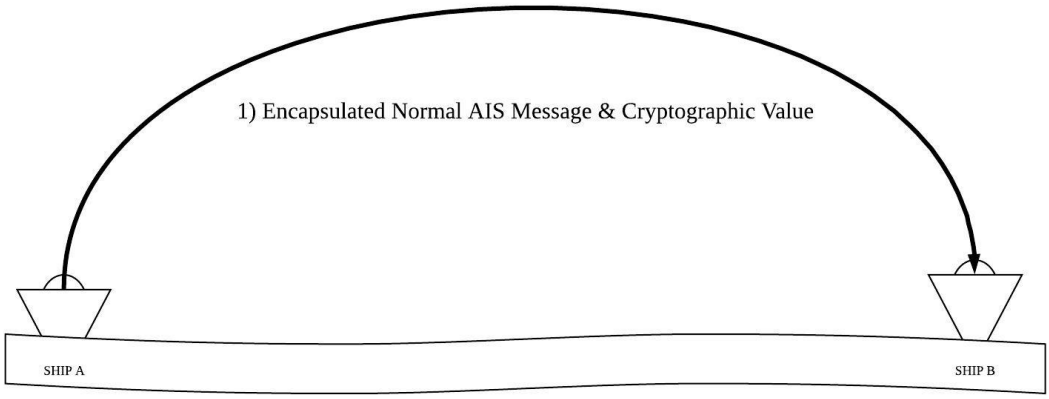


Figure 3.3. Basic visual abstraction of a one-mechanism design architecture.

3.3.2 Two-Mechanism Design Architecture

Figure 3.4 depicts a high-level, visual abstraction of a two-mechanism AIS authenticator design. A two-mechanism architecture features two separate transmissions: 1) the normal AIS data packet and 2) a separate, second AIS data packet encapsulating the cryptographic value of the first normal AIS data packet. For example, if Vessel A intends to transmit Message 1, it would have to transmit Message 1 normally and follow up the initial transmission with a separate AIS data packet that encapsulates the cryptographic value as the AIS message's authentication payload.

The first of two messages adheres to standard AIS data packet transmission procedures. This unaltered AIS transmission would allow receive-only vessels to operate normally and maintain safe situational awareness.

The two-mechanism design architecture is not an optimal design choice because this design

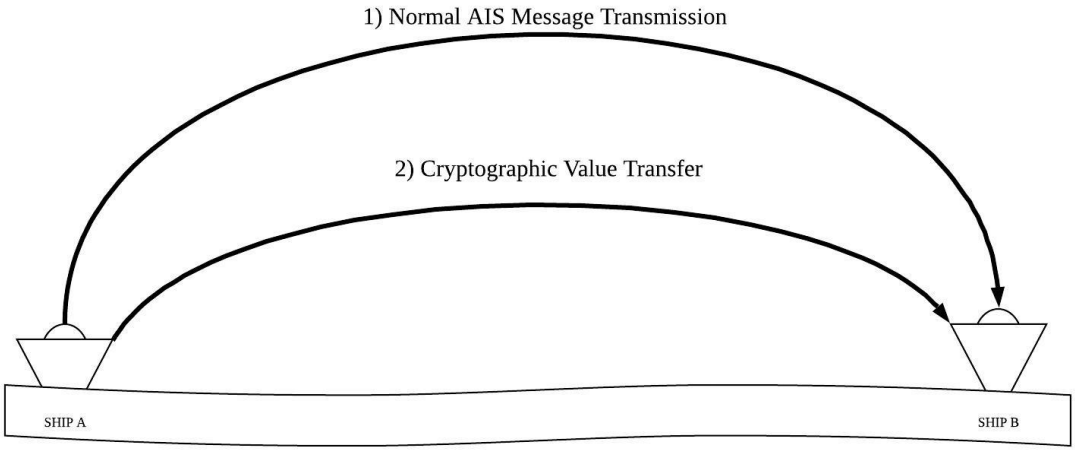


Figure 3.4. Basic visual abstraction of a two-mechanism design architecture.

is inefficient with TDMA; the design wastes limited AIS transmission slots in congested traffic situations and for messages of zero interest.

3.3.3 Three-Mechanism Design Architecture

Figure 3.5 depicts a high-level, visual abstraction of a three-mechanism design architecture. This three-mechanism design features a standard AIS data packet transmission, an interrogation message (challenge), and message authentication response.

As discussed, this first message flow is not special or unique to the current AIS architecture and permits normal operations and maintaining situational awareness.

The second message flow is an interrogation message. A receiving vessel equipped with the an AIS authenticator generates this challenge message, as desired, after receipt of a normal AIS data packet. (During the experiment, the authenticator prototype will be configured to automatically authenticate every received message; this configuration is implemented to measure and assess the prototype's capability of continuous authentication.) The proposed challenge message is a combined encapsulation of the initial normal data packet and a timestamp. The timestamp is measured coarsely so that, in absence of an attacker, it should match the sending timestamp. Although the addition of a timestamp in the interrogation

message does not provide true non-repudiation, the timestamp provides a degree of replay protection.

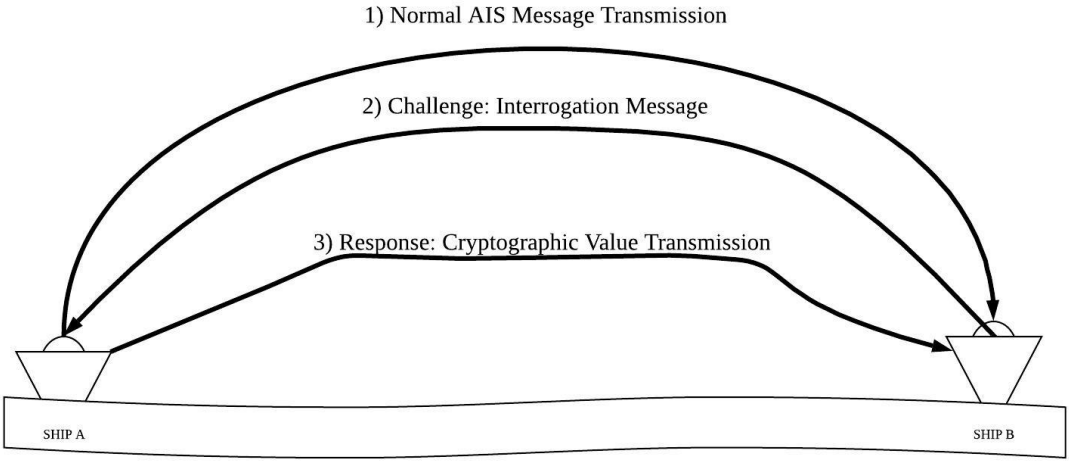


Figure 3.5. Basic visual abstraction of a three-mechanism design architecture.

The third message flow is a response message that incorporates a cryptographic authentication value. A vessel equipped with an AIS authenticator can generate this response message automatically upon receipt of the challenge message. The proposed response message is an encapsulation of a generated authentication cryptographic value over the original AIS data packet. The response message includes the concatenation of a timestamp with the original AIS data packet. This timestamp provides a degree of replay protection and an input the challenging vessel can evaluate against the time that the original AIS data packet was received.

3.3.4 Selecting the Three-Mechanism Design Architecture

This work evaluates the proposed three-mechanism design architecture as requiring the fewest possible message exchanges without violating established message protocols while limiting bandwidth inefficiencies. This three-mechanism design permits normal AIS message transmission, benefiting vessels that are either not equipped with the proposed AIS authentication system or fitted with a receive-only AIS device. In sum, as specified, this three-mechanism design fulfills the following design constraints: compliance with ITU stan-

dards, compliance with NMEA standards, bandwidth economy, and scalability (including receive-only users).

3.3.5 A Detailed Explanation of the Three-Mechanism Design Architecture

With regards to the three-mechanism design, inspiration for the second and third mechanisms originated from military "challenge and response" procedures used in situations when one party encounters another of unknown identity. These challenge and response procedures, using the methodology and three mechanisms illustrated in Figure 3.6, were modified and implemented in our developed AIS authentication algorithm.

Figure 3.6 visually details each mechanism and how the three-mechanism design processes various AIS Messages to incorporate data integrity and authentication. As shown, Ship A records each transmission-ready AIS data packet into a buffer with a timestamp before the data packet's RF transmission because the inclusion of a timestamp provides some protection against replay attacks and various authenticator verification check points for the originator and recipient.

Upon receipt of the initial, standard AIS data packet, Ship B may choose to challenge the legitimacy of the received data packet. This challenge is an encapsulation of the in-question AIS data packet. After challenge message creation, Ship B sends the challenge data packet to Ship A.

Upon receipt of a challenge data packet, Ship A parses the challenge data packet to identify the original AIS transmission then conducts a buffer query to verify the challenger's claim. If the in-question AIS data packet exists within Ship A's buffer, Ship A responds to Ship B with an AIS data packet encapsulating a cryptographic authentication value of the initial, standard AIS data packet and sending timestamp.

However, if this data does not exist within the buffer, Ship A takes no further action. The transmission of a "true-negative" message would require RF bandwidth and computational resources to transmit. In our minimalist approach, this proposed methodology renders the lack-of-response from Ship A sufficient for Ship B to assume data validation error or illegitimacy.

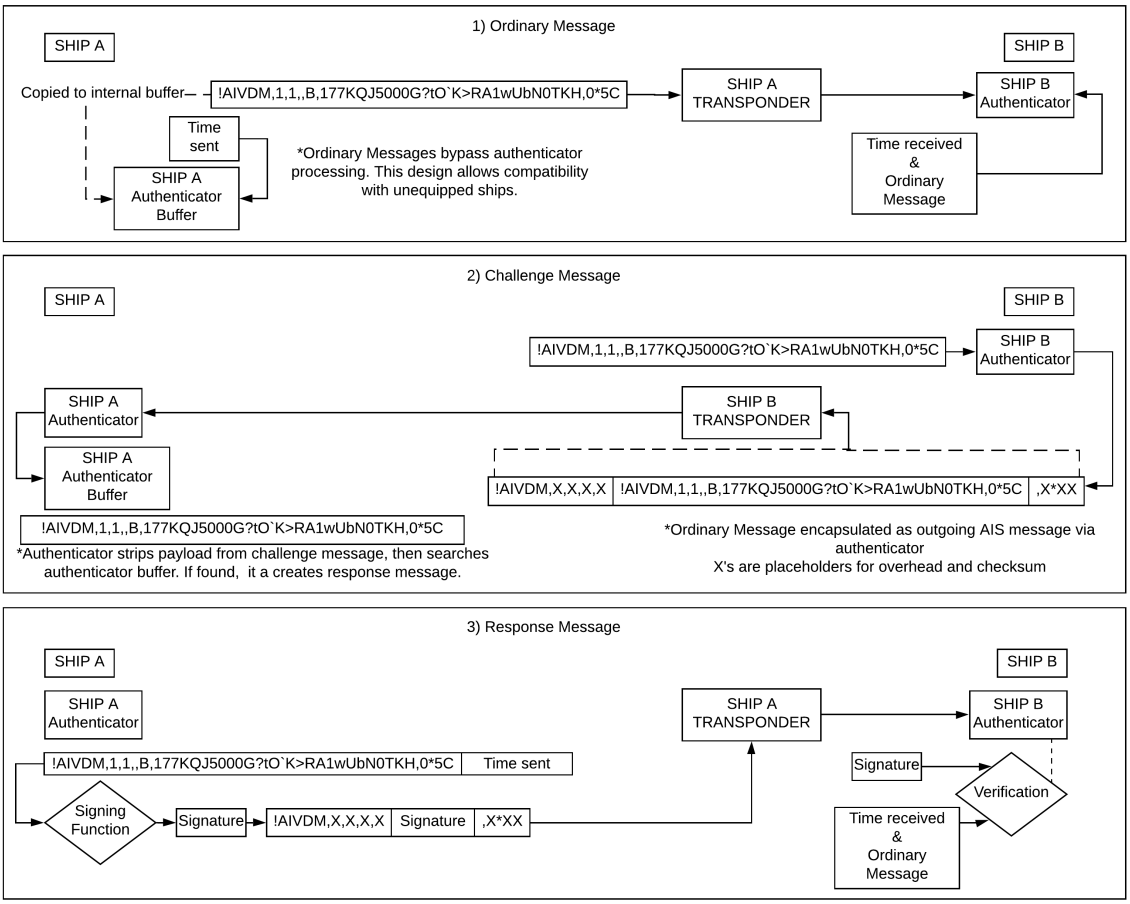


Figure 3.6. Abstraction of the proposed AIS authenticator's functions and mechanisms utilized to challenge and validate an AIS message between two ships.

3.4 Delivery Method

This section examines the proposal to use AIS Message 6 to transmit the challenge message and AIS Message 8 to transmit the response message. Message 6 and Message 8 were ultimately selected because of their payload size, functional flexibility, and designated uses in the current architecture. We begin by evaluating the encapsulation space available in all AIS messages.

3.4.1 Message Types and Available Payload

According to ITU-R M.1371-5, AIS utilizes the concept of a frame that equals one minute. Each frame is divided into 2,250 time slots and each time slot may carry up to 256 bits —the maximum size of one standard AIS data packet [7]. The AIS protocol permits using up to four additional time slots to accommodate more data [7] (i.e., a multi-sentence message. Therefore, the AIS protocol is theoretically capable of transmitting up to 1,280 consecutive bits. Figure 3.7 provides a visual representation of how the AIS protocol fragments large consecutive data packets.

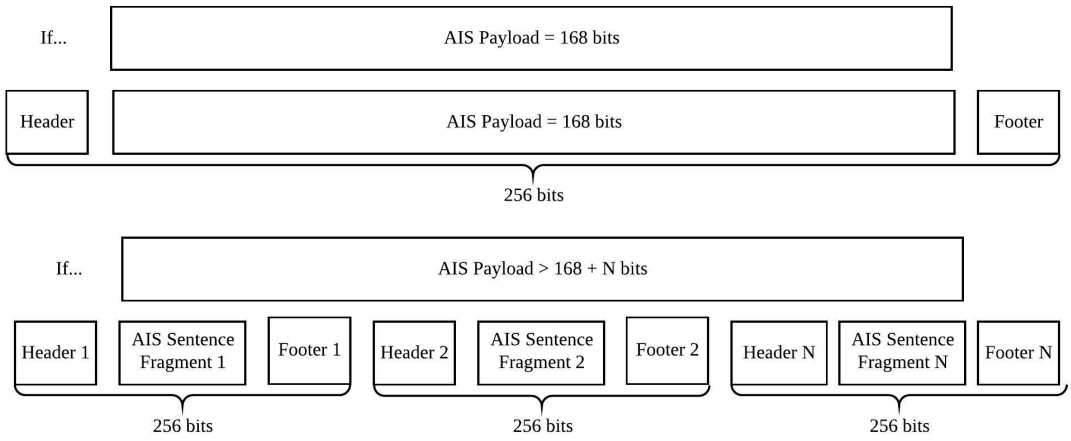


Figure 3.7. Encapsulation and fragmentation for AIS data packets exceeding the 256-bit limit.

However, not every bit within a standard AIS data packet is used for data transmission. Some bits are reserved and allocated for application layer and physical layer overhead. This significantly lowers the number of bits available to encapsulate our proposed cryptographic value. Table 3.2 depicts the various 27 in-use AIS message types and their corresponding maximum payload size given the restrictions discussed in Section 3.2.

A review of the 27 AIS message types reveals that AIS Messages 6 (Binary Addressed Message), 8 (Binary Broadcast Message), 12 (Addressed Safety-related Message), 14 (Safety-related Broadcast Message), and 26 (Multiple Slot Binary Message with communication state) possess the largest possible payload sizes.

Table 3.2. Maximum AIS payloads for specific AIS Message types. Adapted from [7].

Maximum Payload Size for Various AIS Message Types	
Message ID	Maximum Payload Size (bits)
1	168
2	168
3	168
4	168
5	424
6	1008
7	168
8	1008
9	168
10	72
11	168
12	1008
13	168
14	1008
15	160
16	144
17	816
18	168
19	168
20	160
21	360
22	168
23	160
24	168
25	168
26	1064
27	96

3.4.2 Application Specific Messages

ITU-R M.1371-5 [7] states that Application Specific Messages (ASM) are

> AIS messages where the data content is defined by the application are ASM. Examples of this are binary Messages 6, 8, 25, and 26. The data content does not affect the operation of the AIS. AIS is a means for transferring the data content between stations. An ASM's data structure consists of an application identifier followed by the application data.

A 16-bit application identifier uniquely classifies ASM, a feature that could be used to uniquely identify the challenge and response messages in our proposed system. This application identifier is comprised of two parts. The first part is a 10-bit Designated Area Code (DAC) that identifies a specific internationally or regionally managed area. The second part is a 6-bit Function Identifier (FI) that identifies the "data content structure within an application under a DAC assignment" [7]. Table 3.3 is a tabulation and description of the two components included in all ASM.

Table 3.3. A table detailing an application identifier bit 0's through 15's fit and function. Adapted from [7].

Application Identifier	
Bit	Description
15 - 6	Designated area code (DAC). This code is based on the maritime identification digits (MID). Exceptions are 0 (test) and 1 (international). Although the length is 10 bits, the DAC codes equal to or above 1 000 are reserved for future use
5 - 0	Function identifier. The meaning should be determined by the authority which is responsible for the area given in the designated area code

Each DAC may support up to 64 unique Function Identifiers (FIs) [7]; Table 3.4 represents all registered FIs in use. The ITU reserved FIs 33 through 63 for future use.

Table 3.4. Tabular summary of 64 AIS ASM. Adapted from [24].

Function Identifier	**Message Name**
11	Meteorological / Hydrological
12	Dangerous cargo indication
13	Fairway closed
14	Tidal window
15	Extended ship static and voyage-related data
16	Number of persons on board
17	VTS-generated/synthetic targets
18	Clearance time to enter port
19	Marine traffic signal
20	Berthing data
21	Weather observation report from ship
22	Area notice - broadcast
23	Area notice - addressed
24	Extended ship static and voyage-related data
25	Dangerous cargo indication
26	Environmental
27	Route information - broadcast
28	Route information - addressed
29	Text description - broadcast
30	Text description - addressed
31	Meteorological and Hydrographic data
32	Tidal window
33 - 63	Reserved for future use

No application identifiers exist to support our proposed system. Therefore, a custom 16-bit code is necessary.

As we consider message types for our scheme, we note that AIS Messages 6, 8, and 26 have both the longest variable payloads *and* are ASM. We will now evaluate the suitability of

the ASM messages in more detail.

3.4.3 Message 6: Binary Addressed Message

Message 6 is a suitable choice for use as a "challenge" message because an interrogating ship should not broadcast a challenge message as not all vessels in the vicinity may need to see the challenge. As such, a challenge message should address a specific vessel and include a return address (which the interrogated ship will use to respond). The Message 6 format permits fulfilling these two requirements.

AIS Message 6 has two 30-bit source and destination identifier fields and a useful payload limit of 920 bits [7]. The source and destination identifier fields serve a similar purpose as the source and destination fields in TCP. Additionally, this message contains a 16-bit application identifier field. Figure 3.8 is a detailed breakout of the utilization and purpose of each bit in Message 6.

3.4.4 Message 8: Binary Broadcast Message

Message 8 is an optimal choice as the "response" message because of the presence of a source field and lack of a destination field. One might argue that Message 6 could serve as both the challenge and response messages, but Message 8 is a better choice for the following reasons.

A broadcast of the response is more efficient than a single addressed message. There may be other ships in the area that also want to verify the integrity and source authenticity of a particular message. A broadcasted response services multiple vessels, effectively conserving time slots for other users. This is analogous to a school teacher answering one student's question while efficiently responding to several other shy students.

Similar to Message 6, Message 8 also contains the 16-bit application identifier field. However, Message 8 has only one 30-bit source identifier field and a useful payload of 952 bits [7]. The absence of a 30-bit destination field allows for a larger cryptographic value to be transmitted, giving Message 8 a payload limit 32 bits larger than Message 6. Figure 3.9 is a detailed breakout of the utilization and purpose of each bit in Message 8.

Parameter	Number of bits	Description		
Message ID	6	Identifier for Message 6; always 6		
Repeat indicator	2	Used by the repeater to indicate how many times a message has been repeated. 0-3; default = 0; 3 = do not repeat any more		
Source ID	30	MMSI number of source station		
Sequence number	2	0-3		
Destination ID	30	MMSI number of destination station		
Retransmit flag	1	Retransmit flag should be set upon retransmission: 0 = no retransmission = default; 1 = retransmitted		
Spare	1	Not used. Should be zero. Reserved for future use		
Binary data	Maximum 936	Application identifier	16 bits	Bit: 15-6 — Designated area code (DAC). This code is based on the maritime identification digits (MID). Exceptions are 0 (test) and 1 (international). Although the length is 10 bits, the DAC codes equal to or above 1 000 are reserved for future use; Bit: 5-0 — Function identifier (FI). The meaning should be determined by the authority which is responsible for the area given in the designated area code
		Application data	Maximum 920 bits	Application specific data
Maximum number of bits	Maximum 1 008	Occupies up to 3 slots, or up to 5 slots when able to use FATDMA reservations. For Class B "SO" mobile AIS stations the length of the message should not exceed 3 slots For Class B "CS" mobile AIS stations should not transmit;		

Additional bit stuffing will be required for these message types. The table below gives the number of binary data bytes (including application ID and application data), so that the whole message fits into a given number of slots. It is recommended that any application minimizes the use of slots by limiting the number of binary data bytes to the numbers given, if possible:

Number of slots	Maximum binary data bytes
1	8
2	36
3	64
4	92
5	117

Figure 3.8. Detailed breakdown of AIS Message 6. Adapted from [20].

3.4.5 Message 25: Single Slot Binary Message

The ITU-R M.1371-5 primarily utilizes Message 25 for short, infrequent data transmissions [7], [20]. Although this AIS message is capable of addressed and broadcast data applications, its useful payload capacity is limited to 80 bits for addressed applications and 112 bits for broadcast applications. This limit is less than Message 6's 920-bit useful payload and Message 8's 952-bit useful payload. Message 25 is not an optimal choice for the proposed AIS authenticator design because of its smaller useful payload size and designation for short, infrequent data transmissions. Figure 3.10 is a detailed breakout of the utilization and purpose of each bit in Message 25.

Parameter	Number of bits	Description		
Message ID	6	Identifier for Message 8; always 8		
Repeat indicator	2	Used by the repeater to indicate how many times a message has been repeated. 0-3; default = 0; 3 = do not repeat any more		
Source ID	30	MMSI number of source station		
Spare	2	Not used. Should be set to zero. Reserved for future use		
Binary data	Maximum 968	Application identifier	16 bits	Bit 15-6: Designated area code (DAC). This code is based on the maritime identification digits (MID). Exceptions are 0 (test) and 1 (international). Although the length is 10 bits, the DAC codes equal to or above 1 000 are reserved for future use Bit 5-0: Function identifier (FI). The meaning should be determined by the authority which is responsible for the area given in the designated area code
		Application data	Maximum 952 bits	Application specific data
Maximum number of bits	Maximum 1 008	Occupies up to 3 slots, or up to 5 slots when able to use FATDMA reservations. For Class B "SO" mobile AIS stations the length of the message should not exceed 3 slots For Class B "CS" mobile AIS stations should not transmit		

The table below gives the number of binary data bytes (including application ID and application data), so that the whole message fits into a given number of slots. It is recommended that any application minimizes the use of slots by limiting the number of binary data bytes to the numbers given, if possible:

Number of slots	Maximum binary data bytes
1	12
2	40
3	68
4	96
5	121

Figure 3.9. Detailed breakdown of AIS Message 8. Adapted from [20].

3.4.6 Message 26: Multiple Slot Binary Message with Communication State

Message 26 possesses the capability for addressed and broadcast applications and features a useful payload capacity for a 952-bit addressed-variant message and a 984-bit broadcast-variant message. Figure 3.11 is a detailed breakout of the utilization and purpose of each bit in Message 26.

The ITU-R M.1371-5 standard primarily utilizes Message 26 for scheduled binary data transmissions by automated broadcast stations using either the Self Organized Time Division Multiple Access (SOTDMA) or Incremental Time Division Multiple Access (ITDMA) schemes [7], [20]. SOTDMA is the basic RF TDMA scheme for scheduled repetitive transmission from an autonomous AIS station (e.g., shore station broadcasting harbor information). The ITU-R M.1371-5 recommends that SOTDMA should be used by AIS

stations “operating in autonomous and continuous mode” [7]. AIS message types that use the SOTDMA scheme are of “a repeatable character and are used in order to supply a continuously updated surveillance picture to other users of the data link” [7].

<table>
<tr><th>Parameter</th><th>Number of bits</th><th colspan="3">Description</th></tr>
<tr><td>Message ID</td><td>6</td><td colspan="3">Identifier for Message 25; always 25</td></tr>
<tr><td>Repeat indicator</td><td>2</td><td colspan="3">Used by the repeater to indicate how many times a message has been repeated. default = 0; 3 = do not repeat any more</td></tr>
<tr><td>Source ID</td><td>30</td><td colspan="3">MMSI number of source station</td></tr>
<tr><td>Destination indicator</td><td>1</td><td colspan="3">0 = Broadcast (no Destination ID field used)
1 = Addressed (Destination ID uses 30 data bits for MMSI)</td></tr>
<tr><td>Binary data flag</td><td>1</td><td colspan="3">0 = unstructured binary data (no Application Identifier bits used)
1 = binary data coded as defined by using the
16-bit Application identifier</td></tr>
<tr><td>Destination ID</td><td>0/30</td><td>Destination ID (if used)</td><td colspan="2" rowspan="2">If Destination indicator = 0 (Broadcast); no data bits are needed for the Destination ID.
If Destination indicator = 1; 30 bits are used for Destination ID and spare bits for byte alignment.</td></tr>
<tr><td>Spare</td><td>0/2</td><td>Spare (if Destination ID used)</td></tr>
<tr><td rowspan="2">Binary data</td><td rowspan="2">Broadcast Maximum 128
Addressed Maximum 96</td><td>Application identifier (if used)</td><td>16 bits</td><td>
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>15-6</td><td>Designated area code (DAC). This code is based on the maritime identification digits (MID). Exceptions are 0 (test) and 1 (international). Although the length is 10 bits, the DAC codes equal to or above 1 000 are reserved for future use</td></tr>
<tr><td>5-0</td><td>Function identifier (FI). The meaning should be determined by the authority which is responsible for the area given in the designated area code</td></tr>
</table>
</td></tr>
<tr><td>Application binary data</td><td>Broadcast Maximum 112 bits
Addressed Maximum 80 bits</td><td>Application specific data</td></tr>
<tr><td>Maximum number of bits</td><td>Maximum 168</td><td colspan="3">Occupies up to 1 slot subject to the length of sub-field message content

Class B “CS” mobile AIS stations should not transmit</td></tr>
</table>

Figure 3.10. Detailed breakdown of AIS Message 25. Adapted from [20].

Conversely, the ITU-R M.1371-5 recommends that AIS stations use the ITDMA scheme “to pre-announce transmission slots of non-repeatable character” [7] for autonomous and continuous AIS transmissions. The current AIS standard recommends the utilization of

ITDMA (i.e., Message 26) for three occasions: data link network entry, temporary changes and transitions in periodic reporting intervals, and pre-announcement of safety related messages [7]. Essentially, Message 26 ITDMA is reserved for alerting messages vice normal information exchange.

<table>
<tr><th>Parameter</th><th>Number of bits</th><th colspan="4">Description</th></tr>
<tr><td>Message ID</td><td>6</td><td colspan="4">Identifier for Message 26; always 26</td></tr>
<tr><td>Repeat indicator</td><td>2</td><td colspan="4">Used by the repeater to indicate how many times a message has been repeated. 0-3; default = 0; 3 = do not repeat any more</td></tr>
<tr><td>Source ID</td><td>30</td><td colspan="4">MMSI number of source station</td></tr>
<tr><td>Destination indicator</td><td>1</td><td colspan="4">0 = Broadcast (no Destination ID field used)
1 = Addressed (Destination ID uses 30 data bits for MMSI)</td></tr>
<tr><td>Binary data flag</td><td>1</td><td colspan="4">0 = unstructured binary data (no Application Identifier bits used)
1 = binary data coded as defined by using the
16-bit Application identifier</td></tr>
<tr><td>Destination ID</td><td>0/30</td><td colspan="2">Destination ID (if used)</td><td colspan="2" rowspan="2">If Destination indicator = 0 (Broadcast); no data bits are needed for the Destination ID. If Destination indicator = 1; 30 bits are used for the Destination ID and 2 spare bits for byte alignment.</td></tr>
<tr><td>Spare bits</td><td>0/2</td><td colspan="2">Spare (if Destination ID used)</td></tr>
<tr><td rowspan="2">Binary data</td><td>Broadcast Maximum 104</td><td>Application identifier (if used)</td><td>16 bits</td><td colspan="2"><table><tr><th>Bit</th><th>Description</th></tr><tr><td>15-6</td><td>Designated area code (DAC). This code is based on the maritime identification digits (MID). Exceptions are 0 (test) and 1 (international). Although the length is 10 bits, the DAC codes equal to or above 1 000 are reserved for future use</td></tr><tr><td>5-0</td><td>Function identifier (FI). The meaning should be determined by the authority which is responsible for the area given in the designated area code</td></tr></table></td></tr>
<tr><td>Addressed Maximum 72</td><td>Application binary data</td><td>Broadcast Maximum 88 bits Addressed Maximum 56 bits</td><td colspan="2">Application specific data</td></tr>
<tr><td>Binary data added by 2nd slot</td><td>224</td><td colspan="4">Allows for 32 bits of bit-stuffing</td></tr>
<tr><td>Binary data added by 3rd slot</td><td>224</td><td colspan="4">Allows for 32 bits of bit-stuffing</td></tr>
<tr><td>Binary data added by 4th slot</td><td>224</td><td colspan="4">Allows for 32 bits of bit-stuffing</td></tr>
<tr><td>Binary data added by 5th slot</td><td>224</td><td colspan="4">Allows for 32 bits of bit-stuffing</td></tr>
<tr><td>Spare</td><td>4</td><td colspan="4">Needed for byte alignment</td></tr>
<tr><td>Communication state selector flag</td><td>1</td><td colspan="4">0 = SOTDMA communication state follows
1 = ITDMA communication state follows</td></tr>
<tr><td>Communication state</td><td>19</td><td colspan="4">SOTDMA communication state (see § 3.3.7.2.1, Annex 2), if communication state selector flag is set to 0, or ITDMA communication state (§ 3.3.7.3.2, Annex 2), if communication state selector flag is set to 1</td></tr>
<tr><td>Maximum number of bits</td><td>Maximum 1 064</td><td colspan="4">Occupies up to 3 slots, or up to 5 slots when able to use FATDMA reservations. For Class B “SO” mobile AIS stations the length of the message should not exceed 3 slots. Class B “CS” mobile AIS stations should not transmit</td></tr>
</table>

Figure 3.11. Detailed breakdown of AIS Message 26. Adapted from [20].

Given the ITU-R M.1371-5 restrictions on its role and utilization, Message 26 is not an optimal choice for the propose AIS authenticator. The recommended use and application of Message 26 does not support the prioritized function and application of the challenge and response messages. The challenge and response messages from a maritime vessel are not autonomous and periodic, but are dependent upon the cumulative frequency of all received data packets in the maritime environment. This cumulative frequency is not constant and varies in time depending on the environment; this cumulative frequency is not periodic.

Furthermore, the SOTDMA and ITDMA schemes do not easily support implementation of the proposed challenge and response messages. Instead of SOTDMA or ITDMA, the challenge and response messages are more suited for the Random Access Time Division Multiple Access (RATDMA) scheme. The ITU-R M.1371-5 recommends that the RATDMA scheme is used when an AIS station needs to allocate a unannounced time slot [7].

3.5 Choosing between Asymmetric or Symmetric Authentication Schemes

To this point, this work utilized the term "cryptographic authentication value" to avoid any unnecessary bias towards any one of two broad categories of cryptography: asymmetric and symmetric. While it is possible to incorporate both types of cryptography into the novel AIS authenticator design, one is more appropriate for the maritime environment.

Before progressing further, it is important to recognize that these two categories of cryptography cannot be compared on equal footing due to their advantages and disadvantages for specific applications, problems, and the functional environment [23]. Therefore, this research must make the comparison and decision in the context of the constraints and considerations discussed in Section 3.2.

This section does not intend to provide a detailed discussion of each possible asymmetric or symmetric algorithmic variant. However, this section examines the unique characteristics, advantages, and disadvantages of asymmetric and symmetric cryptography with respect to implementation in the proposed system and application in the maritime environment.

3.5.1 Asymmetric Cryptography

In asymmetric cryptography, there is an unprotected public key and a protected private key; the asymmetric system must ensure the security of the private key insofar that private key generation from the public key is infeasible. Thus, the public key can be made public and available to all entities. Anyone with access to the public key can validate the source of a digital signature because only the signatory should have access to the private key.

When used for authentication, asymmetric algorithms refer to digital signatures, where cryptographic keys are used for signing (private key) and verifying (public key) to ensure data integrity and the validity.

There are three satisfactory characteristics that make digital signatures a compelling option for the proposed system:

- Authenticity. The concept of paired cryptographic keys ensure authenticity because only a private key is capable of signing and only the originator has possession of the private key [23], [25]. This also implies unforgeability. If the private key has not been compromised only the signatory has access and knowledge of it [23], [25].
- Uniqueness. The generated digital signature is a unique arithmetic product of the algorithmic function over a specific plain text [23], [25].
- Non-repudiation. Digital signature verification relies on the access to the public key, so the signer is the only entity possessing the private key. Generally, the public key is available to everyone and the verification process is independent of the signature's originator [23], [25].

There are two positive attributes of digital signatures that are complimentary to the maritime environment. First, the use and acquisition of cryptographic public keys do not require prior coordination of participating parties before initial communication at sea. A vessel may take advantage of message integrity and authentication from digital signatures as long as they have access to the corresponding public key, regardless of whether or not the sender and receiver have communicated before. Second, trusted third-parties provide the trust necessary for public key authentication and eliminate the need for vessel-to-vessel trust.

Conversely, there are three unique, negative attributes of digital signatures that are undesirable in the maritime environment. First, to verify a digital signature, a vessel must

possess the corresponding public key. If an underway vessel does not have the public key it must utilize valuable shipboard computational resources and RF bandwidth to acquire the key, typically from the message originator or a shore-based distribution node. Second, the compromise of a private key could have widespread ramifications if all messages are signed using a persistent private key. Third, asymmetric cryptography and digital signatures require significantly more processing power than symmetric cryptography, and signature size is a consideration factor under AIS Message format limitations.

3.5.2 Symmetric Cryptography

Contrary to asymmetric cryptography, symmetric cryptography does not utilize digital signatures for authentication, but instead achieves authentication through use of a Message Authentication Code (MAC). A MAC is a key-dependent one-way function, requiring a shared secret key between both parties [23].

There are four satisfactory characteristics that make MACs a compelling option for the proposed AIS authenticator design:

- Low computational cost. MACs are typically utilize block ciphers or hash functions; therefore, they require fewer computational resources than digital signatures [23], [26].
- Authenticity. Any alterations of the transmitted message are detectable through verification of the authentication tag and message [23], [26].
- Uniqueness. The generated MAC is a unique function over a specific plain text [23], [25].
- Fixed Output Length. Regardless of the size the input message, the generated MAC is always the same length [23], [26].

There are three positive attributes of MACs specific to the maritime environment. First, in the event of a compromised secret shared key, only the data associated with the compromised shared key is affected. Therefore, the effects in the maritime environment are confined only to the two or more vessels that utilized the same compromised secret and do not extend further. Second, MAC generation is computationally less intensive compared to digital signatures in asymmetric cryptography. Finally, the output size of the MAC is attractive due to the space restrictions of the AIS message format.

Conversely, there are two negative attributes of MACs specific to the maritime environment. Namely, key management introduces an administrative overhead that may become too excessive in areas of high-density traffic. If separate key pairs are used for each pair of vessels in the maritime environment, this would introduce scaling and usability issues.

A maritime operating environment of n users would require $\frac{n^2-n}{2}$ distinct shared keys. Maritime key management implies a level of pre-coordination and planning associated with key distribution and usage among various vessels, particularly as the AIS bandwidth is inhibiting for on-the-spot key exchange. This raises potential issues associated with unnecessary bandwidth usage for key coordination and acquisition in unexpected encounter cases, such as two or more vessels unexpectedly encountering each other during passage making.

Second, the disadvantage of a shared secret key is that the risk of a compromised key increases proportionally with the number of users utilizing the secret key; each user has a number of associated threat vectors that affect the cumulative threat calculus of a shared key scenario.

Given these concerns, there are at least two undesirable scenarios that exist within symmetric cryptography that pose a significant disadvantage to either key management or excessive key sharing.

The first scenario involves the maximization of shared symmetric keys among a number of existing entities in an environment. In this case, the number of uniquely shared symmetric keys is maximized and the number of entities sharing these keys is minimized. This situation effectively minimizes the extent of the damage from a compromised key, but creates an environment saturated with unique shared symmetric keys and administrative overhead. It also inhibits message broadcast.

The second scenario is effectively the inverse of the first. In this case the number of uniquely shared symmetric keys is minimized and the number of entities is maximized. This situation allows for reduced administration of key management, but the number of vessels that share the same shared symmetric key is greater. This case creates an environment where all participants must be intensely scrutinized for trust and reliability, due to the increased risk of the secret key being shared to a malicious actor.

3.5.3 Selecting Asymmetric Cryptography

Given the concerns discussed in Sections 3.5.1 and 3.5.2, asymmetric cryptography is the preferred choice for the current AIS architecture and associated challenges of the maritime domain. Specifically, asymmetric cryptography is selected over symmetric cryptography to mitigate against the following unique maritime challenges: limited bandwidth at sea and, very importantly, administrative overhead.

Limited Available Bandwidth at Sea. RF Bandwidth for Internet Protocol (IP) services at sea is a premium commodity. According to [27], the average U.S. broadband internet average bandwidth is 11.7 Mb/s; some area-specific service providers are capable of providing more than 1,000 Mb/s through fiber optic media. Conversely, a typical maritime vessel may have up to 2 Mb/s via a commercial satellite service provider [27]. Due to the limited bandwidth for underway maritime vessels, key acquisition becomes a critical issue and concern for both types of cryptography. One could argue that this concern may be mitigated while vessels are in port and have access to shore-based IP services and capabilities. However, this mitigation is effective only if all vessels encountered while on passage are known and accounted for. This mitigation becomes ineffective when one vessel unexpectedly encounters another vessel and consequently needs ad hoc acquisition of the corresponding cryptographic key. While this issue affects both asymmetric and symmetric cryptography, it is more pronounced in symmetric cryptography. In an asymmetric system, public keys are available to everyone and can be pre-loaded without any ship-to-ship coordination in port; however, shore-based services are unlikely to be able to facilitate key exchange for all potentially encountered underway vessels.

Administrative Overhead. In dense traffic areas, such as Singapore or Dover, the number of unique secret keys would create a significant burden on a vessel's voyage management systems. In an area that experiences 100 vessels per day, there would be 4,950 uniquely shared secret keys (i.e. $\frac{n^2-n}{2}$ keys). For example, [28] reports that Dover Strait sees the passage of approximately 400 ships daily. In this scenario, there would be up to 79,800 uniquely shared secret keys. The selection of an asymmetric cryptographic system over a symmetric cryptographic system eliminates this administrative overhead because of the concept of asymmetric key pairs and the possibility for any receiving vessel to validate the validity of a message using the appropriate public key. Message integrity is facilitated through the utilization of one signing key and one verification key for all users.

3.6 Asymmetric Cryptographic Signature Algorithm Selection

This section discusses various signature algorithm solutions. It is important to focus on the characteristics and technical limitations that may disqualify common algorithms from use in the current AIS standard and the proposed authenticator. This section therefore also considers identity-based and certificateless public key infrastructures as possible alternatives.

3.6.1 Digital Signature Length Limitations to Choosing a Scheme

Regardless of the strengths or benefits of any existing or future cryptographic algorithm or scheme, the hard technical limitation is the AIS standard for message payload size. The selected digital signature algorithm must not generate a digital signature that exceeds Message 8's 952-bit payload limit. In Section 3.4, this research selected Message 6 and Message 8 for the challenge and response messages, respectively. Message 8 will encapsulate the digital signature and its payload limitation is the main discriminator of all possible digital signature algorithms. This payload size constraint significantly narrows the amount and type of eligible digital signature algorithms. Any attempts to increase Message 8's 952-bit payload size require a revision of the ITU-R M.1371-5.

Generally, the bit-level security of a cryptographic algorithm scales according to the size of algorithm's key, modulus, or f in the case of Digital Signature Algorithm (DSA), RSA, and Elliptical Curve Digital Signature Algorithm (ECDSA), respectively [29]. Similarly, the scaling of the bit-wise security level also affects the generated digital signature size. Therefore, the AIS standard limits the maximum possible bit-level security of any eligible algorithm.

In the following comparisons for PKI based signatures, identity-based signatures, and certificateless signatures, we compare on the points of ease of **key distribution, revocation, bandwidth economy, and signature length**.

Computational costs for signature verification and key storage are also considerations with signature algorithm selections [30], but less critical in the AIS context. In particular, modern AIS Class A and Class B maritime vessels possess the computation and transmission capabilities to support the transference and processing of digital signatures.

3.6.2 Public Key Infrastructure (PKI)

This research compares the RSA digital signature scheme [31], ECDSA [32], and DSA [33] within traditional PKI as potential alternatives. These are common digital signature algorithms used in applications such a web computing or email exchange. Elliptic curve variants such as ECDSA are implementations of existing public key algorithms (e.g. DSA) using an elliptic curve [23]. The elements of an elliptic curve over a finite field facilitate the computationally less intensive processing and the faster generation of smaller key sizes [23].

The concept of PKI is applicable and compatible with the maritime environment.

- **Distribution** Each vessel may be treated as an independent party and eligible for both public and private keys. There are trusted international maritime organizations (e.g., IMO, International Association of Marine Aids to Navigation and Lighthouse Authorities (IALA)) that may act as a trusted-third party key certification authority and repository for public keys.
- **Signature Length** National Institute of Standards and Technology (NIST) 800-57 Part 1, Rev 3 is an open source cybersecurity publication published by the National Institute of Standards and Technology, a branch of the U.S. Department of Commerce, and provides recommendations in cryptographic key management guidance. This publication recommends the utilization of cryptographic algorithms that offers at least 112-bit security for applications from 2014 to 2030 [29]. In order for applications to achieve at least 112-bit security, NIST recommends that DSA algorithms utilize a public key of 2048 bits and a private key of 224 bits; RSA utilize a modulus of 2048 bits; and ECDSA an f from 224 bits to 255 bits [29].
- **Bandwidth Economy** While an AIS-equipped maritime vessel is in port, bandwidth economy is not a concern because the maritime vessel would have access to land-based communication methods (e.g., fiber or copper pier-side internet) to acquire all necessary digital certificates of all PKI-enabled maritime vessels. However, when a maritime vessel is underway, bandwidth economy would become a concern for digital certificate acquisition, with the potential consequence of not obtaining a certificate. We recommend, to the maximum extent possible, that maritime vessels acquire all available digital certificates in port in order to minimize bandwidth usage while underway. While underway, bandwidth is utilized to acquire the digital certificates of unexpected maritime vessels. Commercial maritime vessels typically follow a pre-

determined route; this predictability provides insight into which digital certificates are unnecessary while a vessel is underway.

- **Public Key Revocation** A digital certificate and the corresponding public key have a shelf-life before expiry. The Certificate Authority sets an expiration date to each issued digital certificate. However, it is possible for a private key to be compromised before the expiry date, leading to need for revocation. Due to the bandwidth constraints, revocation at sea may be inhibited. Consequently, access to a Certificate Revocation List (CRL) may be limited to time in port, enabling a compromise window of use while the vessel is in transit and unable to access the CRL.

The current AIS protocol only supports a limited payload size. As noted under **Signature Length**, the three common PKI algorithms (i.e., RSA, ECDSA, DSA) have a critical problem: The generated digital signatures are too large for the current AIS protocol to support. Therefore, this research must consider alternative PKI solutions.

Within the maritime environment and AIS protocol is the Maritime Mobile Service Identity (MMSI), an ITU-R M.585-8 standardized vessel identification method. Maritime organizations, local authorities, and vessels utilize the MMSI to identity and publish a vessel's administrative data (e.g., country of origin, registered owner, vessel type, callsign). Any feasible alternative to the three common PKI algorithms should be able to accommodate the MMSI identifier as part of the subject's public values.

3.6.3 Identity-Based Public Key Infrastructure

In 1984, Adi Shamir introduced the identity-based cryptographic scheme [34]. This scheme "enables any pair of users to communicate securely and to verify each other's signatures without exchanging private or public keys, without keeping key directories, and without using the services of a third party" [34]. Shamir explained that the public key is generated from a specific attribute associated with that entity's public identity. Within the maritime community and AIS protocol, the MMSI is a vessel's unique public identifier and thus could be used to calculate an entity's public key for digital signature verification.

Conversely, the "corresponding secret key is computed by a Key Generation Center (KGC)" [34]. This characteristic makes the KGC a possible point of failure in the cryptographic system because a singular third-party entity has knowledge of the private key. Therefore,

trust is particularly necessary for this cryptographic system to function properly.

Goudossis and Katsikas [21] propose a similar solution to what is discussed in this work for securing AIS through identity-based public key cryptography. Goudossis and Katsikas discuss the theoretical application of identity-based public key cryptography providing the following advantages and benefits for identity-based digital signatures:

- **Distribution** The authors state there would not be a requirement for numerous "certificate authorities, sub-certificate authorities, and cross certificate authorities" [21] required by traditional PKI. They suggest that the National Maritime Registers/Authorities may be suited as the trusted third party and therefore simplifying and reducing the infrastructure. Figure 3.12 shows the authors' suggested hierarchy for the identity-based cryptography infrastructure. The authors state that with the absence of multiple certificate authorities, the adoption of this proposed infrastructure would not be necessarily hindered by a typical certificate authority's technical, political, or financial requirements. As an example, note that National Private Keys Generators may co-exist with a root IMO-Private Key Generator.
- **Signature Length** The signature size of an identity-based digital signature is dependent on the specific choice of implementation of bilinear pairing groups based on elliptic curves [35]. For instance, in 2001 Boneh, Lynn, and Shacham proposed an identity-based scheme based on the Weil Pairing that yield signature lengths of 126 bits, 154 bits, 237 bits, 259 bits, and 265 bits with discrete-log bit-wise security of 752 bits, 923 bits, 1417 bits, 1551 bits, and 1589 bits; respectively [35].
- **Bandwidth Economy** An infrastructure utilizing identity-based public key cryptography would not have to consume precious bandwidth for transmission of public keys if such keys are not already loaded in port. Under identity-based PKI, maritime vessels have the ability the generate the public keys of any other registered maritime vessel through the use of the publicly identifiable MMSIs. All valid AIS Messages incorporate a MMSI and would not require additional bandwidth resources to acquire other types of publicly identifiable information to generate a public key.
- **Public Key Revocation** Unlike in PKI, the public key here is the vessel's identity, e.g. MMSI. Identity revocation for an identity-based AIS authenticator system would utilize a CRL, similar to traditional PKI. Using this CRL, a trusted international maritime agency (e.g., IALA, IMO) could then provide the list of invalid MMSIs to

participating vessels [21]. In cases of private key compromise, but still valid MMSI, the vessel can update their private key through contact with the KGC [36].

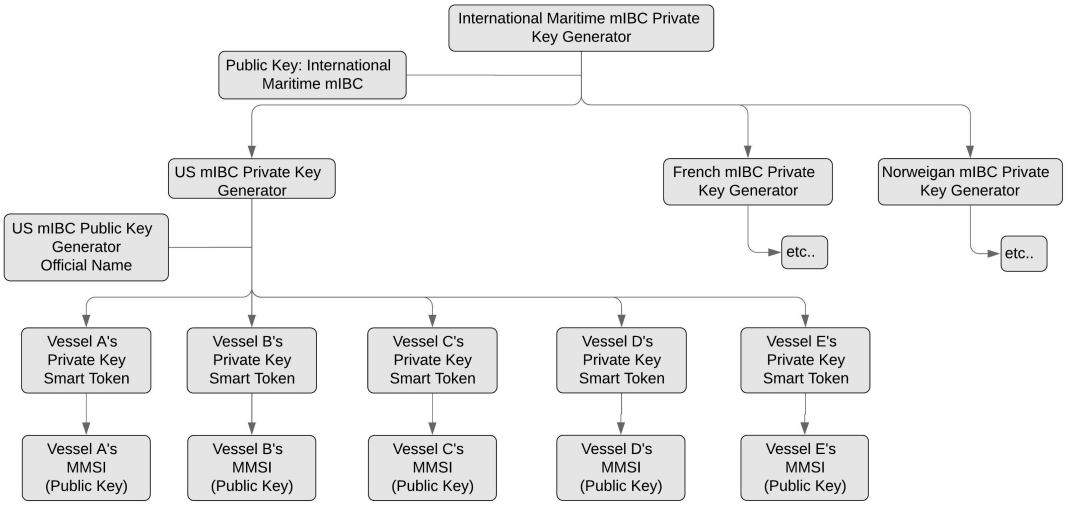


Figure 3.12. Goudossis's and Katsikas's proposed maritime identity-based cryptography architecture. Source: [21].

3.6.4 Certificateless Public Key Infrastructure

In 2003, Sattam Al-Riyami and Kenneth Patterson introduced certificateless public key cryptography [37]. The authors discussed that this certificateless concept relies on the use of a trusted third-party, but "does not require the use of certificates to guarantee the authenticity of public keys" [37]. Certificateless key infrastructures aim to solve some of the key escrow issues of identity-based key infrastructures.

For identity-based public key cryptography, the trusted third-party organization (i.e., the KGC) generates the entity's private key and securely delivers it to that entity. Conversely, anyone may generate the entity's public key from certain aspects of that entity's public information (e.g., MMSI). However, for certificateless cryptography, the public and private key generation process is slightly different. The trusted third-party KGC generates a partial private key from its own master secret key and the entity's public identity [37]. Once the entity received the partial private key from the KGC, the entity will generate its full private

key from a confidential value and the partial private key. With this method, the certificateless concept provides a higher degree of security for the private key than identity-based because the KGC does not have knowledge of the full private key.

Conversely, this system is not identity-based though it possesses similar traits and processes. Specifically, the most significant difference between certificateless and identity-based is the public key is not computable from an identity alone [37]. The entity derives its own public key from the confidential string value and the KGC's public parameters. The public key may be made available to the public either through direct exchange or a publicly accessible database.

Certificateless cryptography provides the following advantages and benefits:

- **Distribution** The certificateless scheme is nearly identical to the identity-based scheme [37] in terms of initial contact with a KGC for obtaining a partial secret key, and mirrors PKI in terms of public key distribution. The proposed AIS system would utilize trusted third party maritime authorities and agencies to simplify and maintain the overall certificateless infrastructure. Within a certificateless framework, an entity's public key is not initially available to others because the public key is derived from the entity's secret information and the KGC's public parameters [37]. However, once the entity generates its own public key it may be available to all other entities via a trusted, secure public repository (such as via in-port loading of public keys) or as attachments in secure messages.
- **Signature Length** Similar to the identity-based scheme, a certificateless digital signature length is dependent on the specific choice of cryptography (see lengths mentioned in Section 3.6.3). Certificateless schemes are constructed from bilinear maps based on groups [37]. In [37], the authors implemented their proposed certificateless scheme using Weil and Tate pairings on elliptic curves [37].
- **Bandwidth Economy** Within the certificateless framework, maritime vessels would not have to consume precious bandwidth for public key acquisition if the public keys of other entities are pre-loaded in port (as discussed for PKI in Section 3.6.2). Typically, this public key pre-loading is a feasible option because maritime vessels often transit the same trade routes like city buses on its daily circuit. Conversely, bandwidth resources are necessary when a maritime vessel needs to obtain the public

key of unexpected vessels while on voyage. In circumstances where another vessel's public key is obtained while at sea, a secure transmission method is necessary to ensure the integrity of the public key. Although the certificateless framework is slightly disadvantaged against the identity-based framework with regards to bandwidth economy, the certificateless framework gains a significant security advantage over identity-based in that the KGC does not have knowledge of an entity's private key because the actual private key is derived from a partial private key and the entity's secret information [37].

- **Public Key Revocation** The certificateless public key revocation methodology would resemble that of the identity-based revocation methodology [37]. The trusted third-party maritime agencies would provide the system users with regularly validated and regulated MMSIs in order to maintain the security of the public-private key pair.

3.6.5 Selecting the Certificateless Public Key Infrastructure

A certificateless public key infrastructure would ensure that our AIS authenticator does not violate the constraints and considerations, discussed in Section 3.2, while also ensuring that the KGC does not have access to all private keys.

A certificateless public key infrastructure would reduce private key vulnerability to the KGC; only the entity has knowledge of its own private key. However, this cryptosystem will require that ships procure the public keys of all possible encounters during its voyage. For the situations involving unexpected contact with another vessel a separate web based public key repository is required to allow ships to acquire any missing public keys not acquired while in port.

3.7 Charm Cryptographic Library: Selection, Utilization, and Limitations

The standard Python library does not provide the necessary commands, functions, and backend processing to support the complex mathematics required for cryptography. Consequently, this research needed a cryptographic library and tool set to process experimental cryptographic values for implementation into the AIS authenticator.

Therefore, this research utilized GitHub, an open source software development platform

and repository, to locate an appropriate Python library that supports the complex computing and processing of various forms of cryptography. The Charm Cryptographic Library [38] is one such tool set.

The Charm Cryptographic Library is a Python-based, collaborative cryptographic tool set and repository for cryptographers and software developers. The Charm Cryptographic Library provides the developer with a range of digital signature algorithms to test for compatibility and potential with the AIS protocol. The Charm Cryptographic Library has adequate support and documentation because its community of developers is fairly active. One significant positive decision point for utilizing the Charm Cryptographic Library is the inclusion of various cryptographic pairing group options, which provide this research with increased flexibility for compatibility determination of certain cryptographic algorithms.

Parings are often used in identity-based and certificateless cryptography, where "pairings map pairs of points on an elliptic curve into the multiplicative group of a finite field" [39]. Within the Charm Cryptographic Library there are a limited number of useful pairing groups available via a Python class object that allows the developer to instantiate a cryptographic scheme under specific pairings:

- Super Singular Curve (SS) 512 or 1024,
- Miyaji, Nakabayashi, and Takano (MNT) 159, 201, or 224,
- Barreto and Naehrig (BN) 254.

This research acknowledges that these six pairing options do not represent an exhaustive list of all possible pairing options and therefore recommend further exploration and implementation of other possible pairings to find an optimal solution in any future research. This research focuses on these built-in implementations to support the proof of concept.

The Charm Cryptographic Library is not frequently updated with new cryptographic algorithms or optimizations of existing ones. Even though there are numerous implementations of various algorithms in the Charm Cryptographic Library, these algorithms are not an exhaustive collection to test for AIS authenticator compatibility, but provide a starting basis. Any existing algorithm not found in the Charm Cryptographic Library must be implemented from scratch and is subject to developer design, coding inefficiencies, and errors.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
A Necessary Alternate Research Methodology

Chapter 3 argued for the development and implementation of an experimental AIS authenticator with a digital signature and identified benefits of using a certificateless approach. The completion of this research requires a feasible implementation of a digital signature algorithm that is compatible with the current AIS and NMEA protocols and standards.

In the following sections, and using the selected open source tools, we walk through our process of identifying a particular scheme for implementation and testing. The Charm Cryptographic Library contains over 50 unique cryptographic schemes and respective Python implementation source code; however, the library does not provide for any certificateless digital signature variant. Thus, while the recommendations of the previous chapters point to a certificateless infrastructure, the immediately available testing tool limits this option for the experimental AIS authentication system.

Consequently, we start this chapter by exploring Chapter 3's second-choice cryptographic implementation and test the scheme [40] for possible compatibility with the proposed authenticator.

As will be shown, among those schemes we were unable to generate an *identity-based* digital signature implementation that did not exceed the current AIS and NMEA data packets' payload size restrictions. Thus we conclude this chapter by discussing the selection and implementation of our third choice Weil-Pairing [35] public key digital signature as a suitable alternative test scheme to further this research.

4.1 Implementation of an Identity-based Digital Signature(IBC_BLS04)

Here we explore our second-best option of an identity-based digital signature implementation as a suitable substitute for the recommended certificateless scheme (our first choice). The research referenced [40] as the primary technical documentation in developing an identity-based digital signature for AIS authenticator testing.

The signature script, *IBC_BLS04.py*, is documented in Appendix M and contains four primary functions: setup(), keygen(), sign(), and verify(). In a true application of this scheme, various entities would access each function at separate instances. However, this script consolidated the four functions into a singular script to facilitate efficient and repetitive iterations of tests and evaluations.

With *IBC_BLS04.py*, the discriminator for usability with the experimental AIS authenticator is the maximum size of the digital signature outputs for each of the available and compatible Charm Cryptography pairing groups: SS512, MNT159, MNT201, MNT224, and BN254.

We tested *IBC_BLS04.py* with the MMSI value "338999999" and the message "HELLO WORLD". The resultant digital signature samples were verified and their respective sizes recorded. The MNT159 pairing group consistently generated the smallest, functional digital signature amongst the four pairing groups.

Table 4.1 displays the maximum size of each digital signature output for the various compatible pairing groups available through the Charm Cryptographic Library. Chapter 5 discusses the significance of Broadcast Binary Messages (BBM) with regard to AIS authenticator digital signature encapsulation and transmission; however, in Chapter 4, the relevant information concerning BBM is its 52-character payload limit. Each BBM character represents a 6-bit ASCII character and the 52-character limit is equivalent to 312 bits or 39 bytes.

Table 4.1. Various maximum sizes of identity-based cryptographic digital signature based on the IBC_BLS04 scheme [40].

IBC BLS04 Digital Signature Size Outputs					
	Available Charm Cryptography Library Pairing Groups				
	SS512	MNT159	MNT201	MNT224	BN254
Maximum Size (bytes)	480	360	412	444	400

The IBC_BLS04 (using MNT159) digital signature is too large for the current AIS and NMEA standards to support; the digital signature size exceeds the 39-byte payload limit for BBM. Even if the digital signature was fragmented and evenly distributed over five consecutive BBM sentences, the 195-bytes cumulative limit is insufficient to support the

360-byte IBC_BLS04 (MNT159) digital signature.

This research caveats that the 360-bytes identity-based digital signature was the result of the IBC_BLS04 and MNT159 combination. There are other pairings available from various sources that may reduce the digital signature size sufficiently to enable an IBC scheme, but the exploration of options external to the Charm Cryptographic Library is outside the scope of this research.

Due to the lack of compatibility and functionality for ready-to-test certificateless and identity-based digital signature schemes, our discussion now shifts to exploring the BLS04 [35] public key signature as a possible third-choice solution for the proposed AIS authenticator.

4.2 BLS04 Digital Signature Selection and Implementation

This research continues by testing the alternate short digital signature scheme based on the Weil-Pairing [35] known as BLS04 after its authors: Boneh, Lynn, and Shacham. Unlike the IBC_BLS04 scheme tested in Section 4.1, BLS04 is not an identity-based digital signature scheme, nor is it a certificateless digital signature scheme. Therefore, it does not possess the unique mitigations against the documented problems of key management, distribution, and administrative overhead of the PKI as discussed in Section 3.6. However, the BLS04 scheme potentially generates a 160-bit digital signature that provides a bit-wise security level similar to the 320-bit DSA [35].

The BLS04 scheme may not be the ideal cryptographic implementation for the proposed AIS authenticator, but this alternative methodology supports and validates the idea for future compatibility and AIS flexibility. Effectively, the proposed BLS04 scheme can serve as a placeholder and simulation for more AIS-compatible digital signature schemes developed in the future.

With regard to the scope of the research and BLS04 implementation, we simulate and simplify the cryptographic infrastructure required through a one-time Python key generation and distribution script. However, a real-world implementation of the BLS04 scheme with the proposed AIS authenticator would require 1) a certified, trusted third-party certificate

authority to validate public keys, 2) a computationally and bandwidth affordable method to acquire public keys while underway or in port, 3) a public key revocation process, and 4) a backend system for vessels to manage and store valid public keys.

4.2.1 Selection Process of BLS04

Within the Charm Cryptography Library, there are seven pre-implemented and available short digital signature schemes for immediate use and testing [35], [41]–[46].

This research acknowledges that these seven schemes are not an exhaustive list of all existing digital signature schemes that may be functional and compatible with the experimental AIS authenticator. Therefore, this research recommends further exploration and implementation of other existing schemes to find an optimal solution. However, due to their availability, the research focused on and explored each of these schemes to determine their viability for the experimental AIS authenticator. The main discriminator for a functional alternative solution (i.e., digital signature scheme and pairing group) is the generated digital signature size.

The BLS04 selection involved collecting data on digital signature sizes while evaluating each of the seven schemes 30 times under every available Charm Cryptographic Library pairing group. Our authenticator was designed to place the received digital signature fragments into a five-position queue which initiates the reassembly of the digital signature fragment for verification when full. Appendix I contains the code for the signature fragment queue.

In the tests, the observed digital signature sizes vary up to 10 percent below the maximum value recorded for a specific combination. Due to this variation, the total number of transmitted digital signature fragments, discussed in Chapter 3, varies between two fragmentation values (e.g., four and five or five and six). Chapter 5 discusses the steps taken to discover the fragment proportions. However, what is relevant to this section is the fragmentation value of five because this value is aligned with and supports the five time slot allowance that was discussed in Section 3.4.

Table 4.2 is a tabulation of the various generated maximum digital signature sizes for each available scheme and pairing group combination.

Table 4.2. Digital signature length comparison utilizing various available digital signature schemes and pairing groups from Charm Cryptography Library. Note that the "blacked-out" portions represent the combinations that resulted in a system fragmentation fault and did not produce a signature.

Digital Signature Length Comparison (in Bytes)							
	Schemes						
Charm Cryptography Library Pairing Groups	[35]	[41]	[42]	[43]	[44]	[45]	[46]
SS512	280	478	288	640	632	636	472
SS1024					1088	1100	796
MNT159	160	244	156	612	500	304	240
MNT201	176	272	176	740	600	348	268
MNT224	184	284	188	792	640	376	284
BN254	188	308	196	640	1148	396	304

The Charm Cryptography Library implementations of the [35] and [42] schemes generated the smallest digital signatures for every pairing group compared to the other five signature schemes. We selected the BLS04 [35] signature scheme as our primary test signature implementation because it generated smaller digital signature sizes for the SS512, MNT224, and BN254 pairing groups.

We determined that the BLS04 and MNT159 pairing group combination consistently generated a digital signature size that varied between four and five fragments. The maximum signature size for four fragments is 156 bytes, but the maximum digital signature size for the BLS04 scheme using MNT159 is 160. As a result, the 10 percent variation causes the total fragment amount to fluctuate causing frequent malfunctions during multiple test iterations.

For a digital signature to qualify for five fragments it must be greater than 156 Bytes but less than or equal to 195 bytes. BLS04 with MNT201, MNT224, and BN254 qualified for five fragments and their size variations did not cause a fragmentation issue. Although longer than MNT159, MNT201 generated a consistent number of fragments —thus reduction in implementation malfunctions due to varying fragmentation. Thus, our proof-of-concept utilized MNT201 because it generates a smaller signature size compared to MNT224 and BN254.

Based on the observations and constraints, the tested design architecture for the AIS authenticator consists of the following: 1) AIS Message 6 and Message 8 as the "challenge and response" delivery method and 2) utilization of the implemented BLS04 Charm Cryptography script derived from [35] with the MNT201 pairing group.

CHAPTER 5:
Experiment Design and Results

Chapter 5 discusses the decisions made to develop the final research methodology and the resultant system performance. Topics include the discovered technical abnormalities; research test plan; the required equipment, tools, and setup; architecture redesigns and mitigations; and the measured metrics of each test phase.

5.1 Experiment Resign due to Equipment Design and Technical Challenges

Section 5.1 discusses the various unexpected findings, technical abnormalities, and modifications to the proposed AIS authenticator design. The topics include CAMINO-701 unsupported named AIS Messages, feasibility of end-to-end testing, and indiscernible Message 12, FA-150 technical deviations from the NMEA standard, and FA-150 inability to transmit six consecutive AIS data packets. End-to-end time is defined as the duration of elapsed time between initial normal AIS data packet transmission and digital signature verification.

5.1.1 ALLTEK Marine AIS Class A CAMINO-701

For an unknown reason, the ALLTEK Marine Class A AIS transponder does not support AIS Message 6 and Message 8, but does support AIS Message 12 and Message 14. Consequently, the proposed design, discussed in Chapter 3, was modified to mitigate this lack of support; Message 12 replaced Message 6 and Message 14 replaced Message 8.

There are no major technical differences between the original and alternate AIS Message types. Message 6 and Message 12 have the same overhead requirements and payload capacity; this is also the case for Message 8 and Message 14. However, there is a functional difference between the original and alternate Message types. Message 12 and Message 14 are Safety Related Messages and do not support DAC and FI codes. The AIS prototype system can function with the alternate AIS Message types, but the DAC and FI codes would

be required in a commercial application of our proposed solution, as discussed later in this chapter.

5.1.2 End-to-End Flexibility and Indiscernible Type 12 Messages.

We desired to conduct an unrestricted end-to-end test to better simulate the live environment. However, a compromise was required to progress the experiments and the authenticator prototype. Ideally, the proposed system would not depend on Message 12 encapsulation, but its own message type (e.g., Message 1, Message 2, Message 3, Message 4) for data encapsulation and transmission.

In the experiment, Message 12 is used to encapsulate the ordinary and challenge messages. The absence of the DAC and FI fields eliminates any practical capability to differentiate between ordinary and challenge messages during an unrestricted end-to-end test.

Without the DAC and FI fields, a receiving AIS transponder will confuse an ordinary message with a challenge message. Effectively, this decision introduces an "indiscernible message" bug into the test environment. This bug results in false positives for both challenge and ordinary messages.

For the case of challenge message false positives, if the receiving transponder initiates the parsing of an ordinary message that was supposed to be a challenge message, the authenticator script will incorrectly parse encapsulated data from an ordinary message. This will result in failed signature validations because the actual and expected strings are different during the hash comparison phase of the signature verification process.

For the case of ordinary message false positives, if the receiving transponder initiates the parsing of a challenge message that is supposed to be an ordinary message, the authenticator script will incorrectly initiate the challenge message creation process and ultimately create a double encapsulated challenge message. This would potentially exceed the message length requirement, cause the AIS transponder device to skip message transmissions, and result in failed signature verification.

This research recommends that follow-on work be performed using fully compliant and late-model AIS devices. Our work was limited to the FA-150 and CAMINO-701 transponder; a requisition order for a late-model FURUNO Class A AIS transponder was submitted, but

the device was not received prior to completion of this research due to substantial lead and delivery times.

Note that a subsequent action following the Message type replacement was the modification of the authenticator script into two specialized scripts specific to a host AIS transponder and this research. Section 5.1.5 discusses this action in more detail.

5.1.3 Abnormality in Published NMEA Technical Standards

Per the NMEA-0183 standard, the maximum number of characters in a NMEA sentence should be 82: two initiating deliminators (i.e.,"$" or "!"), 70 6-bit ASCII characters, and two terminating deliminators (i.e., ⟨CR⟩⟨LF⟩) [8]. Addressed Binary Messages (ABM) and BBM are two types of NMEA Message types.

If multi-sentence AIS data packets (i.e., singular AIS packet transmissions with sequential, fragmented data) are used, the AIS transponder device needs to modify the following three fields in order for the multi-sentence data packet transmissions to be received and reassembled appropriately. First, the "Total Number of Sentences" field indicates the total number of sentences utilized in order to encapsulate the entire multi-sentence message. Second, the "Sentence Number" field indicates the current message's "fragmentation position" in the multi-sentence message series. Third, the "Sequential Message Identifier" field is identical for each message of the same serial; this field is intended to distinguish between consecutively-received multi-sentence message sequence.

Multi-sentence ABM data packets must be formatted so that the "first sentence may contain up to 48 valid Six Bit codes" and the "following sentences may contain up to 60 valid Six Bit codes" [8]. Conversely, multi-sentence BBM data packets must be formatted so that the "first sentence may contain up to 58 valid Six Bit codes" and the "following sentences may contain up to 60 valid Six Bit codes" [8].

We conducted a preliminary test to verify the FA-150's transponder device's ability to construct multi-sentence data packet transmissions. This auxiliary experiment was necessary to certify that the authenticator's coding and construction supported the digital signature fragmentation, transmission, and reconstruction sub-functions. The FA-150 was the primary test transponder.

This auxiliary experiment involved the construction of an eight part BBM multi-sentence data packet transmission, which contained a cumulative payload of 400 random NMEA-0183 ASCII characters to represent a possible response message. The auxiliary test script divided the 400-character payload into multiple 58-character sets and modified the corresponding multi-sentence fields in accordance with the NMEA-0183 standard for correctness of BBM Message construction.

The resulting sentence series was created and passed to the transponder for transmission. The FA-150 accepted the sentence series, but it did not transmit the consecutive data packets. This was confirmed via the transmitting transponder's diagnostics menu and the lack of an auditory alert from the receiving transponder. Multiple tests were performed on the FA-150 transponder, but each failed.

An assessment of the auxiliary test script was conducted for syntax and function errors. We referenced [8] for a possible technical solution but were unable to determine the cause of error. Follow-on troubleshooting steps involved revisiting Maritec's online ABM/BBM NMEA Message encoding tool [47] to formulate known, valid NMEA and AIS Message types of various lengths to confirm the cause of failure.

The initial set of sample sentences was a Message 14 type encapsulating a rudimentary "HI" data string. The first test message was copied and pasted into the auxiliary test script for transmission and the test ended in success. Successive attempts to find the cause of failure involved incremental concatenations of the "HI" string to "HIHI", "HIHIHI", and et cetera until the point of failure was encountered. The subsequent tests revealed that the FA-150 transponder device's BBM limit is 52 characters vice 58.

The next subsequent auxiliary experiments were designed to verify that the FA-150 multi-sentence data packet transmission capability functions and operates in accordance with the NMEA-0183 standard. First, we crafted a two-sentence data packet for fragmentation and consecutive transmission. We modified the multi-sentence message fields, and passed both to the transponders but this first test failed. Second, we tried a modified approach utilizing a 50-character sentence. We constructed the sentence as a two-sentence multi-sentence message (two messages each containing a 25-character payload and sent in sequential order). This second test failed as well. Third, we utilized the same 50 character message, but crafted it to be two singular messages (no modification of the multi-sentence fields) and

passed it to the transponder. The third test was successful.

The research surmised that the issue was related to how the multi-sentence fields are utilized. We reviewed [8] again, but did not find any clear technical or procedural information detailing the process governing multi-sentence message construction. Other possible reasons for this abnormality are that FURUNO did not follow the recommendations published in [8] and/or FURUNO uses a 52-character design as a form of anti-reverse engineering. Due to time and functional necessity, we did not test for this abnormality on the CAMINO-701 because the CAMINO-701 functioned as expected with regards to the receiving and sending of specific Message types used in this research.

Next, the research tested the transmission of the original 400 character sentence as seven separate single-sentence data transmissions. In a fourth auxiliary experiment, the test script was modified to transmit the single sentences at 1 second intervals per specifications [8]. The test was a success, but only the first five sentences were transmitted. This discovery led to the next issue regarding the FA-150's buffer limitations.

5.1.4 Limitations on Consecutive Data Packet Transmissions

The research hypothesized that the FA-150 AIS transponder was only able to transmit five consecutive data packets due to its internal buffer limitations.

A fifth auxiliary experiment attempted to consecutively inject valid single-sentence messages to the FA-150 transponder device. The research consistently observed that the transponder would transmit packets number one through five and eight, but fail packets six, seven, and nine. The research surmised that there must be a manufacturer design implementation that clears the buffer state after a certain amount of time or number of transmitted data packets relative to the buffer state. Similar tests were not performed on the CAMINO-701 because it was not essential to the completion of the experiment.

The research did not pursue this issue further because this unforeseen design abnormality did not severely impact our experiment or the overall script. The FA-150's five-data-packet buffer limitation was aligned with our original five consecutive data packet technical constraint discussed in Chapter 3.

5.1.5 Authenticator and Experimental Redesign

Due to the discussed issues and technical abnormalities, we modified our authenticator script to continue the experiment. Consequently, we omitted specific functions from *Challenger.py* and redesigned the single script into two specialized scripts: *A_Challenger.py* and *B_Challenger.py*. The CAMINO-701 would utilize A_Challenger to accept ordinary data packets and send challenge messages; this modification avoids the false positive scenarios discussed previously because A_Challenger would only have to listen for ordinary (Message 12) and response messages (Message 14). Conversely, the FA-150 would utilize B_Challenger and would exclusively listen for challenge messages (Message 12).

This modification limits the experimental possibilities of conducting unrestricted end-to-end testing such as the simulation of the continuous transmissions of two maritime entities and the potential measurement of maximum system load before failure. However, this modification was essential to continue the research and experiments. Additionally, it must be noted that this research will continue to utilize "*Challenger.py*" when referring to the authenticator scripts in order to emphasize the theoretical feasibility of a fully integrated script to generate challenge and response messages. Distinctions between *A_Challenger.py* and *B_Challenger.py* are made when necessary.

The source code for *A_Challenger.py* is included in Appendix G and its process flow graph is included in Appendix H. Similarly, the source code for *B_Challenger.py* is included in Appendix I and its process flow graph is included in Appendix J.

5.2 Testing Plan

This section discusses the one administrative phase (Phase 0) and four operational phases (Phase 1 - 4) that comprise the overall research testing plan. The testing plan incorporates all necessary mitigations and modifications discussed in previous sections. Additionally, this section introduces the term AIS VHF Data-link Message (AIVDM), a NMEA-0183 sentence designed to transfer the entire contents of a received AIS data packet, on the VHR Data Link, to various other maritime vessels.

5.2.1 Phase 0: Key Generation Authority and Simulated Secure Private Key Distribution

The *KeyGenerationAuthority.py* script, documented in Appendix C, was designed to simulate and simplify all the necessary backend administration and processing involved with private and public key creation, management, distribution, and certification. J. Ayo Akinyele, a Charm Cryptography Python primary contributor and collaborator, developed the cryptographic portions of the source code for *KeyGenerationAuthority.py* based on the Boneh-Lynn-Shacham Identity Based Signature [40].

As discussed in Chapter 4, this research utilizes the BLS04 scheme [35] to mitigate previously discussed issues and provide validation for a proof-of-concept.

Phase 0 provides each entity (Ship A and Ship B) with their respective private keys and each other's public keys. This necessary initial phase is required for an operational system. The design and implementation of a trusted third-party entity to provide the necessary backend administration is outside the scope of this research but is commonly available and used in industry.

5.2.2 Phase 1: User-defined NMEA Message Injections, Transmission and Receipt of AIVDM

The Phase 1 criteria for success are the successful injection, acceptance, conversion, transmission, and receipt of any user-defined data packet from one experimental AIS authenticator to the other. Specifically: The FA-150 AIS transponder and authenticator pair will transmit converted user-defined data packets to the CAMINO-701 AIS transponder and authenticator pair.

Verification of Phase 1 success involves checking the CAMINO-701 device's receive logs and message alerts; when an AIS transponder receives a data packet it generates an audible and visual alert. This is the definitive method to accurately verify that an AIS transponder received an AIS data packet. Figure 5.3 shows an example visual alert on the CAMINO-701 display.

During Phase 1, we utilized *InputEncoder.py*, documented in Appendix E. *InputEncoder.py* is a custom Python3 script designed for text file parsing, conversion, and encoding. Ap-

pendix F documents the script's process flow. *InputEncoder.py* executes the following functions:

- Reads from a user-generated text file containing 50 lines of arbitrary, iterative string data (e.g., "ONE," "TWO," "THREE"). For the experiment, stringdata.txt is the designated text file.
- Converts each line of string data from 8-bit ASCII into its corresponding 6-bit NMEA-0183 ASCII representation. See Appendix B for a detailed explanation of the conversion scheme involved with a typical AIS data packet.
- Generates an ABM embedded with the converted string data as its payload. The ABM will manifest in the following format:
  "!AIABM,x,x,x,xxxxxxxxx,x,x.x,s...s,x,*hh⟨CR⟩⟨LF⟩". Note: s...s is the arbitrary string data payload converted to 6-bit NMEA-0183 ASCII sentence.
- Passes the ABM to the AIS transponder for automatic AIVDM conversion and immediate RF transmission.
- Records the outgoing AIVDM and transmission timestamp into an output file for the purpose of verifying data packet transmissions in later phases. For this experiment, Outfile.txt is the designated outfile. This designated file contains all the previously sent data packets from the arbitrary string data text file.

The FA-150 will initiate the start of Phase 1 with the execution of the *InputEncoder.py* script. Phase 1 concludes with the FA-150's successful transmission of a normal user-defined AIS data packet.

5.2.3 Phase 2: Creation and Transmission of Challenge Message

The CAMINO-701's successful receipt of the AIVDM initiates Phase 2. Phase 2 criteria for success are the CAMINO-701's successful creation and transmission of a challenge message derived from the FA-150's output AIVDM in Phase 1.

Phase 2 involves the CAMINO-701 receiving the AIVDM from the FA-150 transponder, the relaying of the received AIVDM to the CAMINO-701's experimental AIS authenticator, the creation of a corresponding challenge message, and the transmission of the corresponding challenge message to the FA-150. Verification of successful challenge message creation is through the *A_Challenger.py's* diagnostic print statements. Verification of successful

AIVDM transmission is through the CAMINO-701's diagnostics menu and transmission logs.

The relaying of the initial AIVDM to the CAMINO-701's authenticator requires the use of the serial.readline() Python function. Conversely, the creation of a challenge message requires the use of *Challenger.py*. *Challenger.py* is a multi-purpose application for challenge and response message generation. However, because of the issues discussed in previous sections, the CAMINO-701 uses *A_Challenger.py* to actively listen for incoming AIVDM from the FA-150 transponder and to generate the corresponding challenge message. *A_Challenger.py* parses the AIVDM, records the time received, converts, and encapsulates the entirety of the received AIVDM within a new ABM. After the creation of the challenge message, *A_Challenger.py* routes the ABM to the CAMINO-701 AIS transponder device for transmission to the FA-150.

5.2.4 Phase 3: Receipt of Challenge Message, Digital Signature Creation, and Transmission

Phase 3 makes the first implementation of cryptography in this test. Previous phases were primarily concerned with the adaptation and integration of the authentication system's software and hardware with the two AIS standards' electro-physical constraints and manufacturer design limitations. Phase 3 criteria for success are the FA-150's successful receipt of the challenge message originating from the CAMINO-701, the parsing and referencing of the challenge message's payload against the log of previously transmitted data packets within the designated outfile, the FA-150 authenticator's automated decision to either create a digital signature or not based on the results of the outfile check function, and the transmission of the response message containing the digital signature to the CAMINO-701.

The FA-150's receipt of the challenge message from the CAMINO-701 initiates Phase 3. *B_Challenger.py* intercepts the received AIVDM and parses the AIVDM for the in-question data packet, which *A_Challenger.py* encapsulates in the previous phase. B_Challenger references the in-question data packet against the array of sent data packets in the designated outfile. When *B_Challenger.py* references the outfile, it searches for the in-question AIVDM and automatically determines whether to create a response message or not based on the data packet's existence in the outfile. Note that for each test iteration, the designated outfile

contains every ordinary data packet created during Phase 1 as a controlled variable.

When the *B_Challenger.py* locates the in-question data packet within the outfile, it extracts the data packets' stored timestamp (when the data packet was sent during Phase 1) and signs the concatenation of that timestamp and AIVDM as the response message's payload. The resulting sentence manifests in the following format: "!AIBBM,x,x,x,x,x.x,s...s,x,*hh⟨CR⟩⟨LF⟩". Lastly, *B_Challenger.py* forwards the resultant digital signature to the FA-150 transponder device for transmission to the CAMINO-701. If *B_Challenger.py* is not able to locate the data packet within the outfile, it prints a status message and take no further action. The test protocol measures this outcome as a failure via a negative performance mark.

B_Challenger.py was not programmed to transmit a "fail" message to the CAMINO-701 in the event of an error in response message creation. The first reason for a lack of response is unnecessary redundancy in experimental error reporting of the experimental AIS authenticator's code. Second, a negative response is a suitable minimalist approach to reducing bandwidth usage and establishing an implicit "fail" response between the CAMINO-701 and FA-150. Other researchers may implement a "fail" response function; however, "fail"-response implementation is not aligned with the research constraints discussed in Chapter 3.

Verification of results is provided by monitoring the CAMINO-701 for successful receipt of the response messages and the FA-150 authenticator's diagnostic Python print statements of the resulting digital signatures and response messages prior to transmission.

5.2.5 Phase 4: Receipt and Verification of Digital Signature

Phase 4 is the final experimentation phase of this research. The successful completion and validation of Phase 4 demonstrates that the current AIS protocol has sufficient flexibility to add integrity and authenticity to AIS data packets without introducing changes to the current standards.

Verification of successful digital signature receipt and verification are the final steps of *A_Challenger.py*'s signature verify function. This verification function will either return an indication of success or failure.

Phase 4 criteria for success are the successful receipt of the response message originating from the FA-150, the parsing of the digital signature payload, and verification of the digital signature.

5.3 Experimental Setup

We have now defined the test phases and criteria for success. Section 5.3 discusses all equipment, tools, and libraries used during this research. This includes all important configuration settings and connections between each experimental device.

5.3.1 Equipment and Tool List

The following is an itemized list of the all equipment and tools utilized.

- Figure 5.1 provides a basic visual representation of the overall experimental connections and setup.
- ALLTEK Marine CAMINO-701 Class A AIS Transponder, as shown in Figure 5.2
- ALLTEK Marine Junction Box, as shown in Figure 5.3
- FURUNO FA-150 Class A AIS Transponder, as shown in Figure 5.4
- Three RS422 Serial to USB Converters, as shown in Figure 5.5
- Faraday Cage, as shown in Figure 5.6. Figure 5.7 provides a closer look at the ports used to connect each AIS transponder units via a coaxial wire.
- Two laptops with at least two free USB ports each
- Two Ubuntu Linux Virtual Machines: 1) 1.90GHz single-core processor assigned to FA-150 transponder, 2) 2.0 GHz single-core processor assigned to CAMINO-701.
- Two RF Antennas
- Python3 Programming Language
- Charm Cryptography - Python3 module and library
- Stanford University's Pairing-Based Cryptography Library
- Python3 "Subversion" library (Charm Cryptography dependency)
- Python3 "M4" library (Charm Cryptography dependency)
- Python3 "python3-setuptools" library (Charm Cryptography dependency)
- Python3 "python3-dev" library (Charm Cryptography dependency)
- The GNU Multiple Precision Arithmetic library

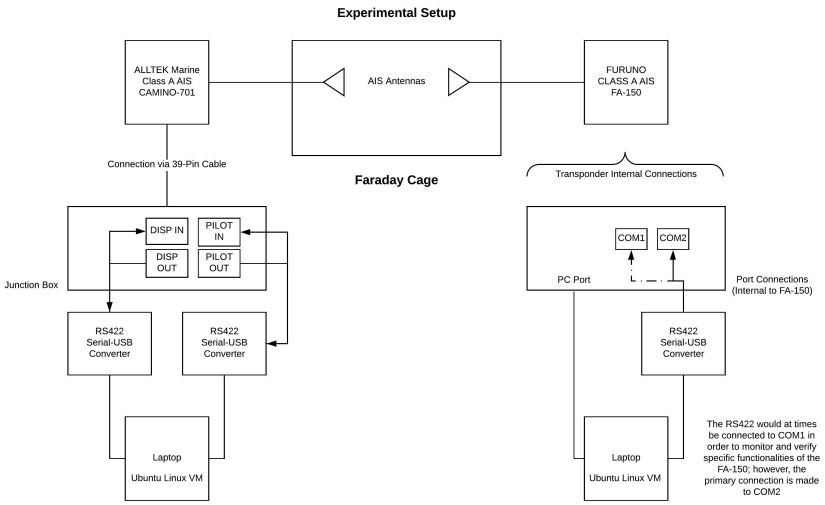


Figure 5.1. Basic outline of the layout and connections of all experiment tools and devices.

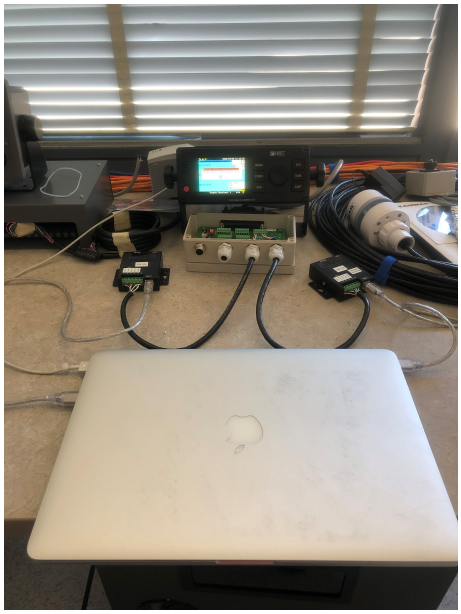


Figure 5.2. ALLTEK Marine: CAMINO-701 Class A AIS Transponder, Junction Box, and Laptop.

5.3.2 Experimental Configuration, Connections, and General Setup

Figure 5.4 shows the basic setup for the FA-150 AIS transponder, its supporting tools, and devices. Each of the laptop's two free USB ports were connected to the FA-150 either directly via the PC port or indirectly via the RS422 adapter to COM1/COM2.

Figure 5.3. Junction Box and the "DISP" and "PILOT" serial connection ports.

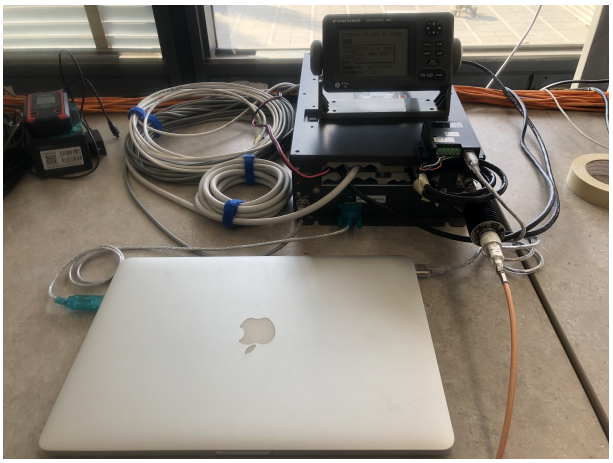


Figure 5.4. FURUNO FA-150 connected via RS422 to Operator Laptop

Figure 5.8 and Figure 5.9 document a close up depiction of the connections to the FA-150 AIS transponder while Figure 5.10 is the wiring diagram for COM1 and COM2 (located under the bottom cover of the FA-150 AIS transponder). Only wiring points one through five were utilized for both COM1 and COM2. Specifically, the AIS "TX" wiring points

were connected to the RS422's "RX" wiring points and vice versa. Ground was connected to ground for both devices.

Figure 5.5. Close up depiction of the RS422 Serial-to-USB converter.

Figure 5.6. The Faraday cage enclosure utilized to house the two RF antennas in order not to interfere nearby maritime traffic in Monterey Bay.

The FURUNO installation manual provides information regarding which NMEA Message types each port supports. Figure 5.11 details one of the port information pages. The COM1, COM2, and PC ports support the input of an ABM and BBM as well as the output of VHF Data Message (VDM)s. This fact influenced the decision to utilize the COM1, COM2, and

PC ports instead of the other available FA-150 ports.

The FA-150 serial port connections utilize a number of different device mode settings depending on the specific application or device in use. This research utilizes either EXT DISPLAY or STANDARD mode to transmit either ABMs or BBMs. Serial ports COM1, COM2, COM3, and COM4 support EXT DISPLAY mode and are NMEA Input and Output Message capable. Similarly, the PC serial port is capable of both NMEA Input and Output Messages while in STANDARD mode.

Figure 5.7. A closer look at the ports used to connect the two coaxial wires from the AIS transponders to the RF antennas inside the Faraday cage.

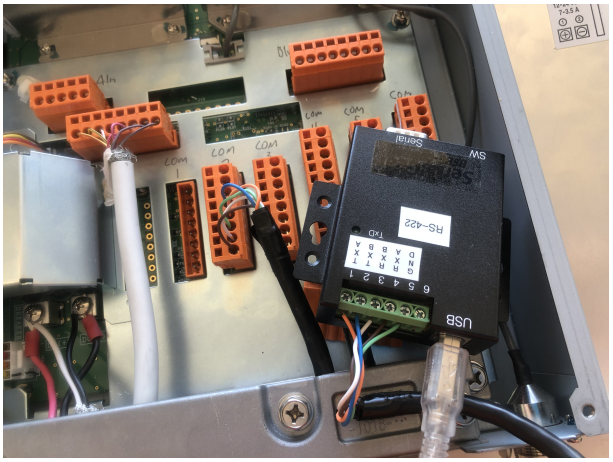


Figure 5.8. FA-150 serial COM ports. Only COM1 and COM2 were utilized.

For the experiment, NMEA sentence injects occurred on PC (STANDARD mode). The

monitoring of the AIVDM occurred on both COM1 (EXT DISPLAY mode) and COM2 (MONITOR) to test and verify the differences and functionality of EXT DISPLAY and MONITOR.

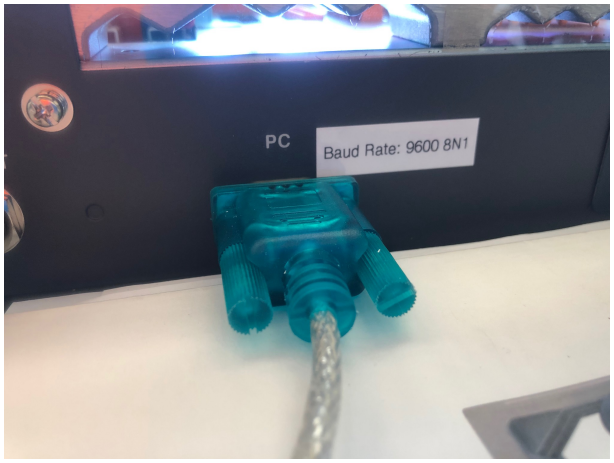


Figure 5.9. FA-150 PC port.

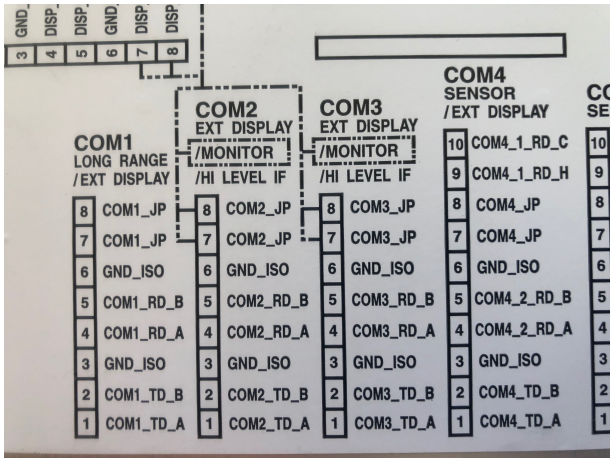


Figure 5.10. FA-150 wiring diagram for COM1 and COM2 found underneath the removal cover. Utilized to connect to serial side of RS422 adapter.

The following is the FURUNO FA-150 transponder's pertinent configuration settings and information:

- MMSI: 338999999
- TX power: 1W

- COM1 port mode: EXT DISPLAY; speed: IEC 61162-2
- COM2 port mode: MONITOR; speed: IEC 61162-2
- PC port mode: STANDARD; speed: 38400 baud

Figure 5.2 shows the basic setup for the ALLTEK Marine CAMINO-701 transponder, its supporting tools, and devices. Each of the laptop's two free USB ports were connected to the CAMINO-701 indirectly via the RS422 adapter to DISP IN/OUT and PILOT IN/OUT within the junction box. The junction box was connected to the AIS transponder unit via a 39-pin parallel port cable.

Figure 5.3 and Figure 5.12 detail how the wires are connected from both RS422 to each of the IN/OUT positions of DISP and PILOT ports.

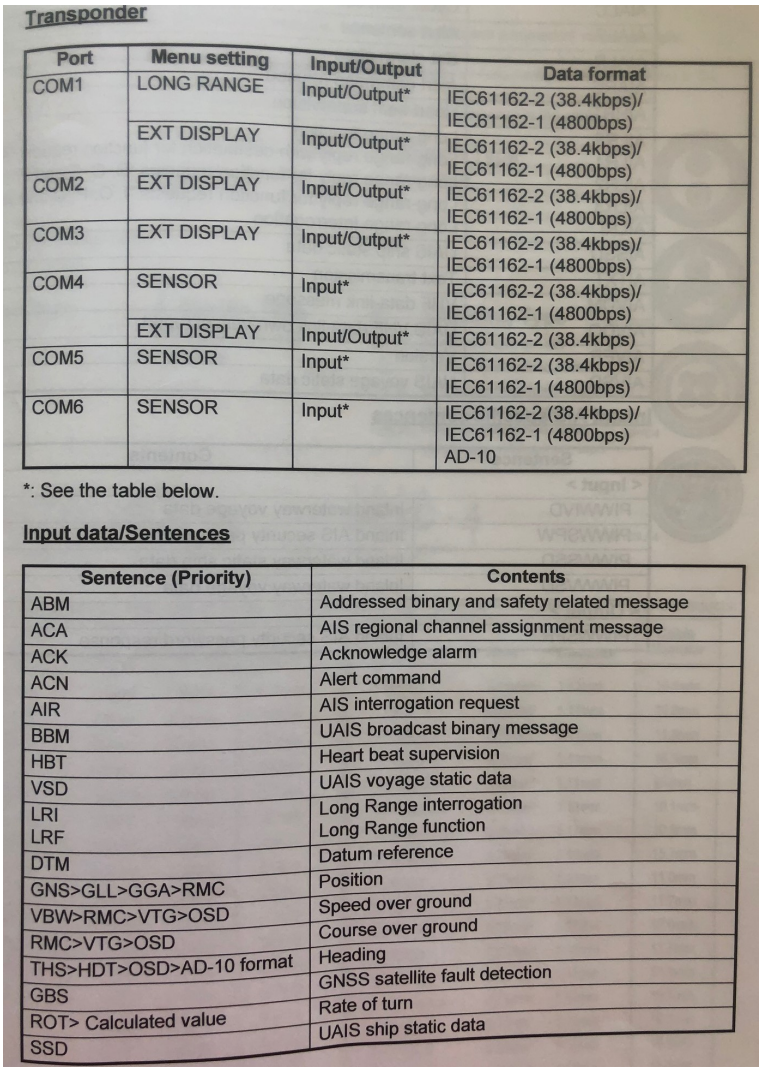
Unlike the FURUNO FA-150, the CAMINO-701 does not have many ports that support ABM, BBM, and AIVDM. The ALLTEK Marine installation manual, as shown in Figure 5.13, states that only the DISP and PILOT ports are capable of supporting ABM, BBM, and AIVDM; the tests used the DISP and PILOT ports for this reason. The NMEA sentence injects occurred on the CAMINO-701's PILOT port while the monitoring of outgoing AIVDMs occurred on the DISP port.

The following is the ALLTEK Marine CAMINO-701 Class A AIS transponder's pertinent configuration settings and information:

- MMSI: 366123000
- TX power: 1W
- DISP port baud rate: 38400
- Pilot port baud rate: 38400

5.4 Procedures, Results, and Phase Completion

Section 5.4 discusses the completion of the four operational phases and includes details regarding the methodology, required inputs, and output data of each phase's completion.



Transponder

Port	Menu setting	Input/Output	Data format
COM1	LONG RANGE	Input/Output*	IEC61162-2 (38.4kbps)/ IEC61162-1 (4800bps)
	EXT DISPLAY	Input/Output*	IEC61162-2 (38.4kbps)/ IEC61162-1 (4800bps)
COM2	EXT DISPLAY	Input/Output*	IEC61162-2 (38.4kbps)/ IEC61162-1 (4800bps)
COM3	EXT DISPLAY	Input/Output*	IEC61162-2 (38.4kbps)/ IEC61162-1 (4800bps)
COM4	SENSOR	Input*	IEC61162-2 (38.4kbps)/ IEC61162-1 (4800bps)
	EXT DISPLAY	Input/Output*	IEC61162-2 (38.4kbps)
COM5	SENSOR	Input*	IEC61162-2 (38.4kbps)/ IEC61162-1 (4800bps)
COM6	SENSOR	Input*	IEC61162-2 (38.4kbps)/ IEC61162-1 (4800bps) AD-10

*: See the table below.

Input data/Sentences

Sentence (Priority)	Contents
ABM	Addressed binary and safety related message
ACA	AIS regional channel assignment message
ACK	Acknowledge alarm
ACN	Alert command
AIR	AIS interrogation request
BBM	UAIS broadcast binary message
HBT	Heart beat supervision
VSD	UAIS voyage static data
LRI LRF	Long Range interrogation Long Range function
DTM	Datum reference
GNS>GLL>GGA>RMC	Position
VBW>RMC>VTG>OSD	Speed over ground
RMC>VTG>OSD	Course over ground
THS>HDT>OSD>AD-10 format	Heading
GBS	GNSS satellite fault detection
ROT> Calculated value	Rate of turn
SSD	UAIS ship static data

Figure 5.11. FURUNO FA-150 technical manual depicting the supported NMEA Input and Output Messages specific to the available serial port connections. Source: FURUNO FA-150 Installation and Operation Manual.

5.4.1 Phase 1 Completion: User-defined NMEA Message Injections, Transmission, and Receipt of AIVDM

Prior to conducting Phase1, both laptops were loaded with *A_Challenger.py*, *B_Challenger.py*, and *InputEncoder.py*. The laptop connected to the CAMINO-701 required the utilization of *A_Challenger.py*. The laptop connected to the FA-150 also required *InputEncoder.py* and *B_Challenger.py*. After the initial staging of the three Python3 scripts, the following

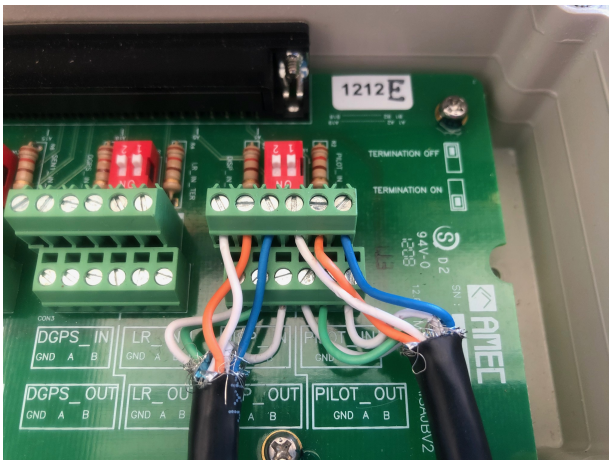
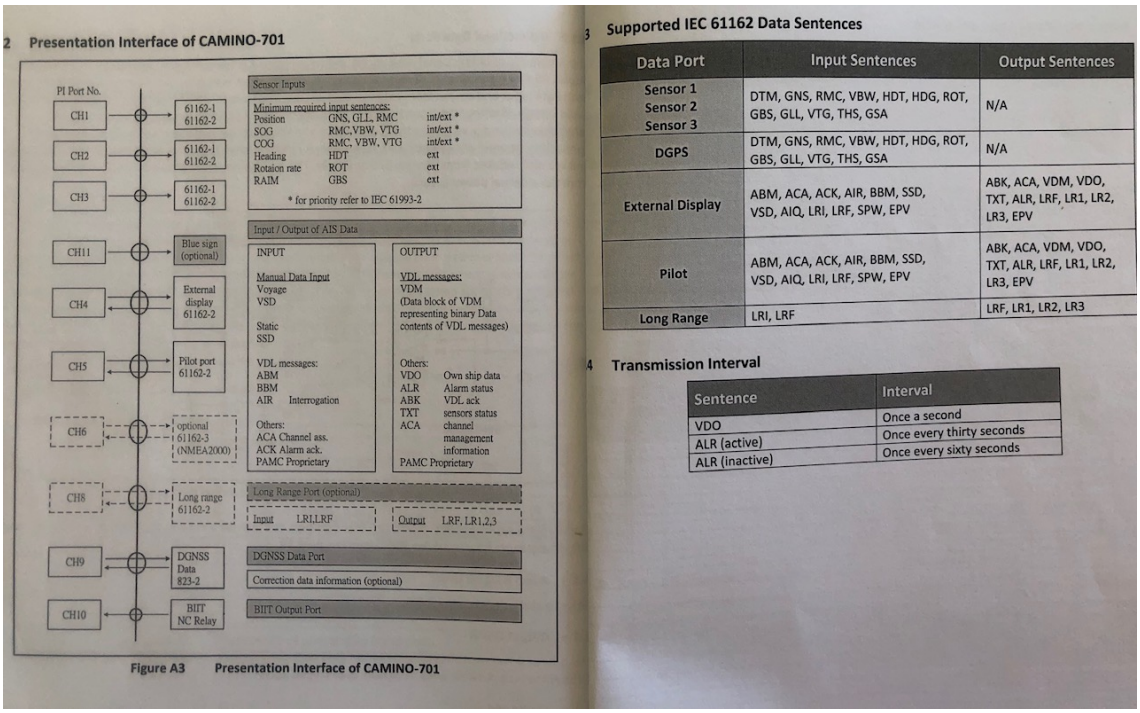


Figure 5.12. A closer view of the CAMINO-701 junction box.



2 Presentation Interface of CAMINO-701

Figure A3 Presentation Interface of CAMINO-701

3 Supported IEC 61162 Data Sentences

Data Port	Input Sentences	Output Sentences
Sensor 1 Sensor 2 Sensor 3	DTM, GNS, RMC, VBW, HDT, HDG, ROT, GBS, GLL, VTG, THS, GSA	N/A
DGPS	DTM, GNS, RMC, VBW, HDT, HDG, ROT, GBS, GLL, VTG, THS, GSA	N/A
External Display	ABM, ACA, ACK, AIR, BBM, SSD, VSD, AIQ, LRI, LRF, SPW, EPV	ABK, ACA, VDM, VDO, TXT, ALR, LRF, LR1, LR2, LR3, EPV
Pilot	ABM, ACA, ACK, AIR, BBM, SSD, VSD, AIQ, LRI, LRF, SPW, EPV	ABK, ACA, VDM, VDO, TXT, ALR, LRF, LR1, LR2, LR3, EPV
Long Range	LRI, LRF	LRF, LR1, LR2, LR3

4 Transmission Interval

Sentence	Interval
VDO	Once a second
ALR (active)	Once every thirty seconds
ALR (inactive)	Once every sixty seconds

Figure 5.13. ALLTEK Marine CAMINO-701 technical manual depicting the supported NMEA Input and Output Message specific to the available serial port connections. Source: ALLTEK Marine CAMINO-701 Installation and Operation Manual edition 1.14.

commands were used to run each script:

For *A_Challenger.py*:

- "sudo python3 *A_Challenger.py* -mmsi 366123000"

For *B_Challenger.py*:

- "sudo python3 *B_Challenger.py* -mmsi 338999999"

For *InputEncoder.py*:

- "sudo python3 *InputEncoder.py* -f stringdata.txt -sm 338999999 -dm 366123000 -o Outfile.txt"

The authenticator script was designed and programmed to require minimal user input. the "-mmsi" Python argument is necessary to synchronize all of the authenticator's functions to the paired AIS transponder.

For *InputEncoder.py*, "-sm" represents source MMSI and "-dm" represents destination MMSI. "-sm" was passed 338999999 to represent the FA-150 AIS transponder and "-dm" was passed 366123000 to represent the CAMINO-701 AIS transponder. These two Python arguments are required to synchronize the *InputEncoder.py* script to the two AIS transponder in the test environment. The purpose of *InputEncoder.py* is to simulate a generic application on a vessel's integrated maritime voyage network, which passes data to the AIS transponder for transmission.

A_Challenger.py and *B_Challenger.py* may be executed simultaneously or sequentially, but these two scripts must run prior to executing *InputEncoder.py*. If *InputEncoder.py* was executed before the two authenticator scripts, the CAMINO-701 authenticator's listening function would not receive the first ordinary data packet transmission and would have to wait one minute before the next ordinary data packet transmission from the FA-150.

Phase 1 goals were successfully achieved. The FA-150 authenticator, via *InputEncoder.py*, parsed the designated text file stringdata.txt and successfully converted the 8-bit ASCII data strings into its corresponding NMEA-0183 6-bit ASCII equivalent. After 6-bit ASCII conversion, the data string was encapsulated into properly formatted ABM and AIVDM.

There are two reasons for this test. First, the FA-150 transponder device requires an ABM

Input Message to generate an AIVDM. Second, it was necessary to certify the accuracy of the *InputEncoder.py*'s data conversation and encoding functions.

The certification process involved comparing *InputEncoder.py*'s generated AIVDM with the received AIVDM from the CAMINO-701. The test certified that *InputEncoder.py*'s data converting and encoding functions are reliable and accurate.

After the generation of the ABM, the FA-150 AIS authenticator forwarded the ABM to the FA-150 transponder device for transmission to the CAMINO-701.

Phase 1 concluded successfully and there were no noteworthy events or new abnormalities.

Figure 5.14 and Figure 5.15 show python print notifications for *A_Challenger.py* and *B_Challenger.py*, respectively, during its active listening state. Figure 5.14 displays three "heartbeat" data packets from the FA-150 transponder. These "heartbeat" data packets are independent of this research and are a manufacturer-programmed feature. In these two figures, *A_Challenger.py* and *B_Challenger.py* did not take additional actions and continues its active listening loop because the two transponders did not detect any conditional data packets (i.e., Message 12 and Message 14).

Figure 5.16 is a screenshot of *InputEncoder.py*'s various parsing, converting, and encoding subfunctions to transform an 8-bit ASCII data string into a 6-bit ASCII ABM and AIVDM.

Figure 5.17 depicts a sample of *InputEncoder.py*'s generated AIVDM stored in the designated outfile. Note that each AIVDM was stored with a corresponding timestamp of when the AIVDM was transmitted from the FA-150 to the CAMINO-701.

The designated outfile was used to validate the accuracy of *InputEncoder.py*. The AIVDM Messages in the outfile was compared to the received AIVDMs on the CAMINO-701.

5.4.2 Phase 2 Completion: Creation and Transmission of Challenge Message

The research transitioned immediately into Phase 2 with the successful completion of Phase 1. Upon receipt of the AIVDM from the FA-150, the CAMINO-701 transponder immediately forwarded the data packet to the authenticator. *A_Challenger.py* parsed the received AIVDM and generated an appropriate challenge message for transmission to the

FA-150.

Figure 5.18 is a screenshot of various portions of Phase 2 via *A_Challenger.py*'s print notifications. The first notification shows that the CAMINO-701 received the Message 12 from the FA-150. The second and third notifications show the initiation of the data packet analysis and challenge message creation process. The final notifications show the various steps and subfunctions required to generate the challenge ABM that was encapsulated with the in-question string. Later, this ABM was routed to the CAMINO-701 transponder for automated conversion into an AIVDM for RF propagation to the FA-150.

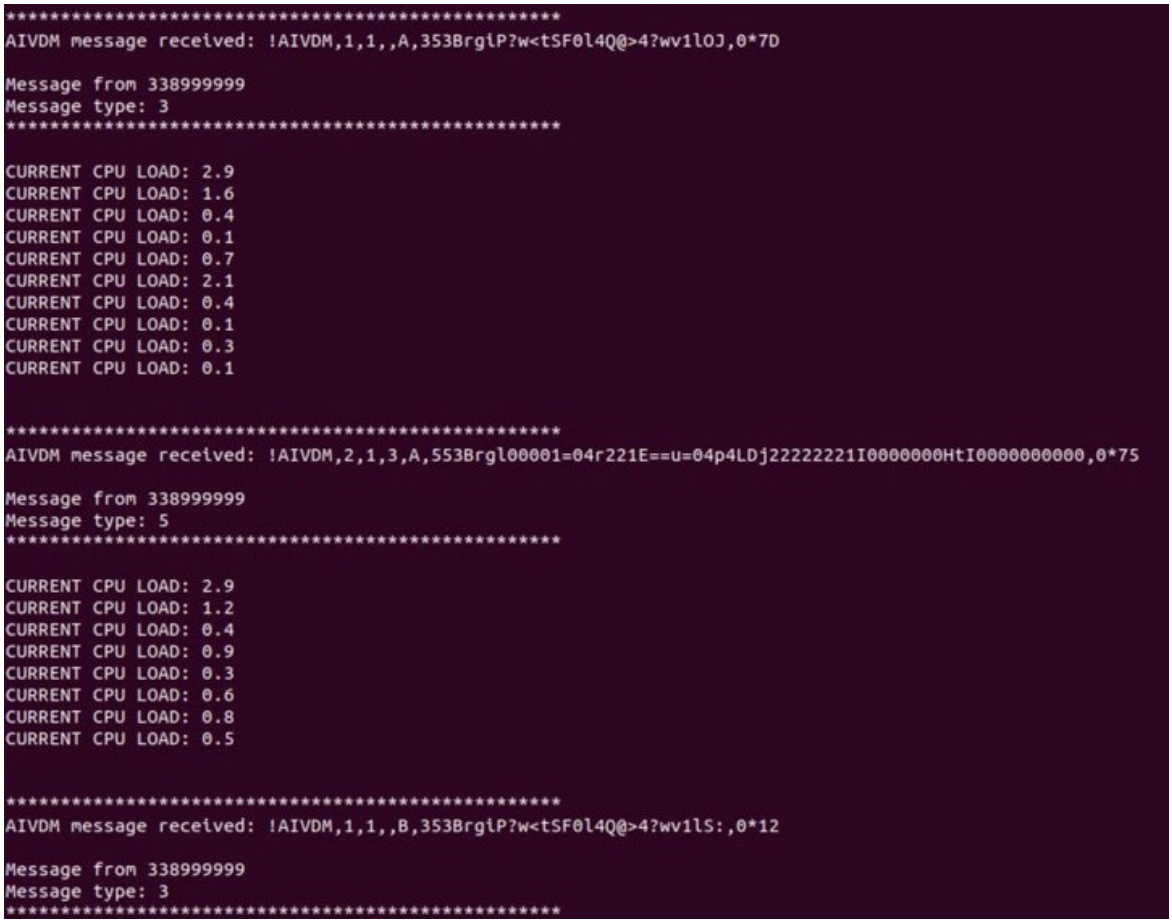


Figure 5.14. CAMINO-701 (laptop) display of received AIS data packets from FA-150 transponder.

With regards to the actual payload of *A_Challenger.py*'s generated challenge message, it is important to discuss the necessity of the *A_Challenger.py*'s double conversion function.

Commas (,) are not represented in the NMEA 6-bit ASCII alphanumeric scheme, but are represented in the AIS 6-bit ASCII alphanumeric scheme. It is not possible to directly pass a AIS 6-bit ASCII formatted AIS data packet to transponder for transmission because all data passed to the transponder for transmission must be in the NMEA 6-bit ASCII format. This process is detailed in Appendix B.

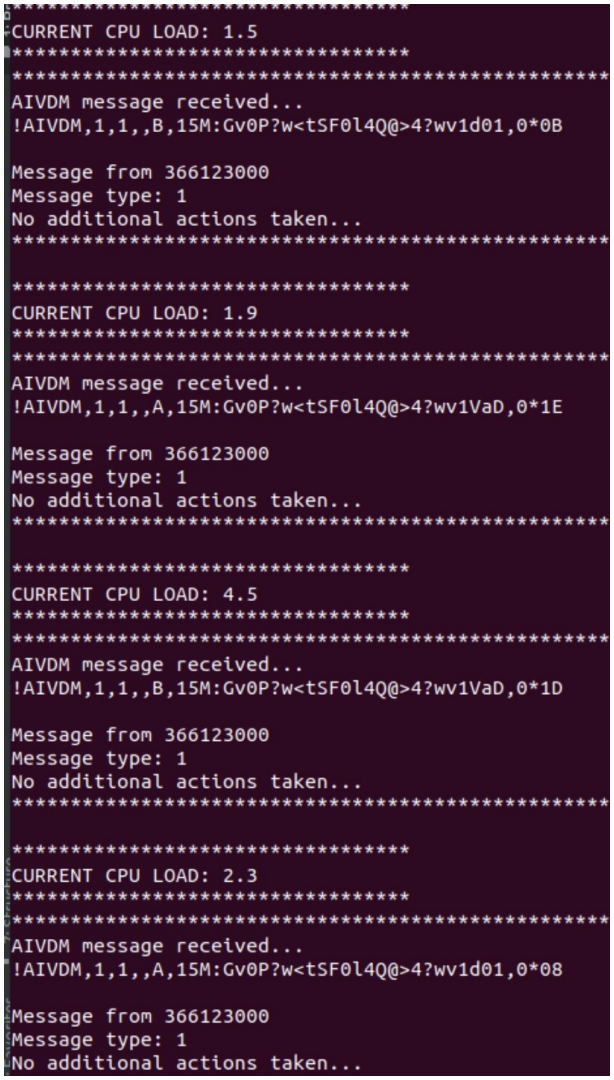


Figure 5.15. FA-150 (laptop) display of received AIS data packets from CAMINO-701 transponder.

Double conversion is the process of converting AIS 6-bit ASCII to binary then to NMEA 6-bit ASCII. The double conversion function provides the CAMINO-701 transponder with

a properly formatted ABM sentence for transmission.

Phase 2 concluded as expected and there were no additional noteworthy events or abnormalities. Phase 2 goals were successfully achieved. Reviewing the CAMINO-701's transmission logs as well as the FA-150's message receive logs verified the success of Phase 2 goals. The CAMINO-701 transponder and authenticator pair received the initial AIVDM, generated a challenge message, and sent it to the FA-150. Similarly, the FA-150 transponder authenticator pair received the challenge message from the CAMINO-701.

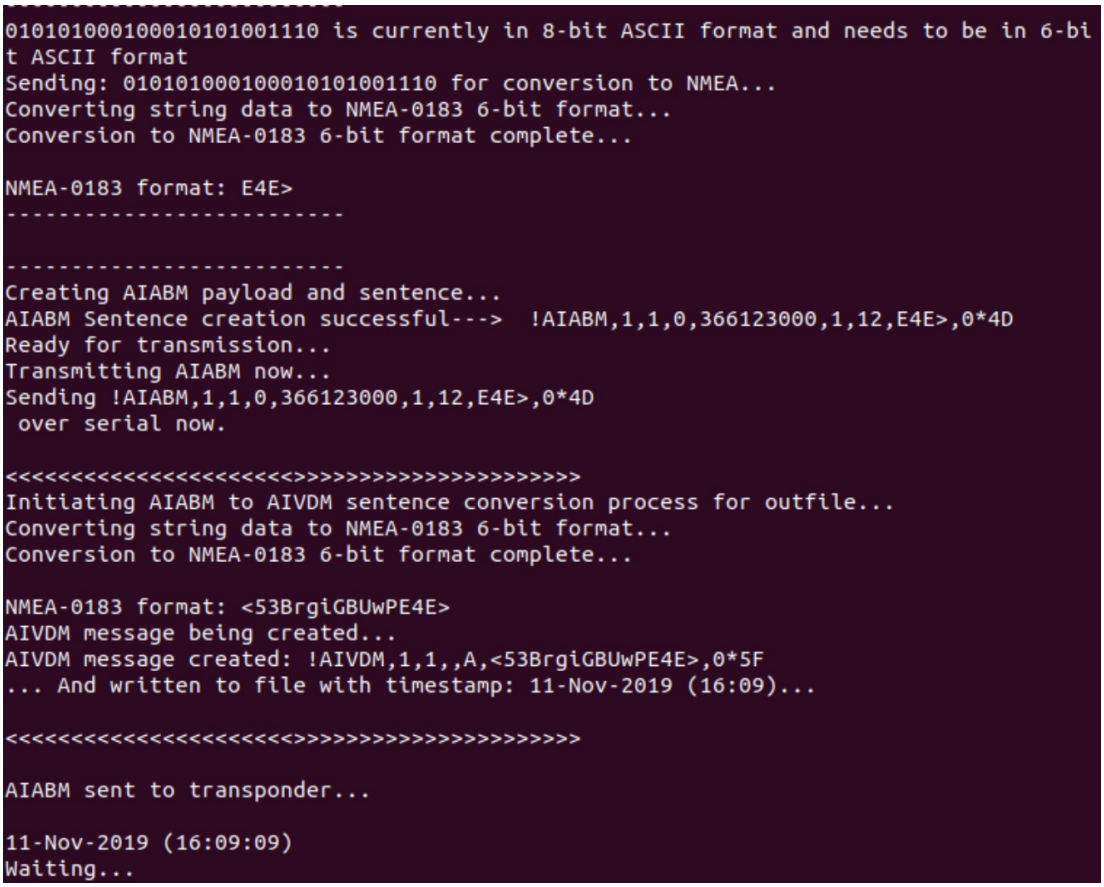


Figure 5.16. A display of *InputEncoder.py*'s process readouts regarding the data inputs, conversions, and outputs.

5.4.3 Phase 3 Completion: Receipt of Challenge Message, Digital Signature Creation, and Transmission

Upon receipt, *B_Challenger.py* parsed the received challenger message from the CAMINO-701 for information such as the MMSI, AIS Message type, and original AIVDM (Phase 1). Once *B_Challenger.py* acquired the requisite information, it initiated the digital signature generation and BBM creation processes.

Figure 5.19 is a screenshot of the various steps required to generate and transmit five digital signature fragments. The first print notification represents the instance when the FA-150 authenticator received the challenge message from the CAMINO-701. The second notification depicts *B_Challenger.py*'s extraction of the MMSI and payload information, which feeds the subsequent digital signature creation and transmission steps.

```
11-Nov-2019 (16:59)|!AIVDM,1,1,,A,<53BrgiGBUwPClq5,0*3E
11-Nov-2019 (17:00)|!AIVDM,1,1,,A,<53BrgiGBUwPE5M?,0*57
11-Nov-2019 (17:01)|!AIVDM,1,1,,A,<53BrgiGBUwPE4QBADD,2*74
11-Nov-2019 (17:02)|!AIVDM,1,1,,A,<53BrgiGBUwPATuEDP,4*60
11-Nov-2019 (17:03)|!AIVDM,1,1,,A,<53BrgiGBUwPATUFA@,4*56
11-Nov-2019 (17:04)|!AIVDM,1,1,,A,<53BrgiGBUwPDlUH,0*60
11-Nov-2019 (17:05)|!AIVDM,1,1,,A,<53BrgiGBUwPDlEFADp,2*09
11-Nov-2019 (17:06)|!AIVDM,1,1,,A,<53BrgiGBUwPADU7B5@,2*07
11-Nov-2019 (17:07)|!AIVDM,1,1,,A,<53BrgiGBUwPCTU>A@,4*2C
11-Nov-2019 (17:08)|!AIVDM,1,1,,A,<53BrgiGBUwPE4E>,0*5F
```

Figure 5.17. A partial display of Outfile.txt. This shows a sample of the data strings converted from simple 8-bit ASCII to final AIVDM sentences that are expected on the CAMINO-701 receiving end.

The digital signature generation process utilized the Charm Cryptography Library and the Weil pairing cryptographic signature scheme discussed in Chapter 4. J. Ayo Akinyele developed the Weil pairing scheme from [35] and implemented it for use with the Charm Cryptography Library. This scheme was exercised independently multiple times prior to performing test Phases 0 - 4 to validate the reliability of the implemented scheme and the authenticator's digital signature verification functions. These tests were completed without any issues or errors.

Since the generated digital signature was larger than the FA-150's 52 character BBM payload limit, *B_Challenger.py* fragmented the resultant digital signature into five fragments and

were transmitted to the CAMINO-701. Phase 3 concluded as expected without errors or abnormalities and Phase 3 goals were successfully achieved. Verification of success involved reviews of the FA-150 transponder's and CAMINO-701 transponder's received messages logs. For the FA-150, the review involved a positive confirmation of receipt for the CAMINO-701's challenge message. For the CAMINO-701, the review involved a positive confirmation of receipt for the FA-150's five transmitted digital signature fragments.

Figure 5.20 is a screenshot of the five Python printouts on the CAMINO-701 that represents the receipt of the response messages, which contains the fragmented digital signatures.

5.4.4 Phase 4

This phase certified the success of *A_Challenger.py*'s digital signature verification function by reviewing the Python print notifications for positive indicators and the lack of negative indicators (e.g., Python script crash errors due to verification failures).

When the CAMINO-701 received the response messages (Message 14) from the FA-150, *A_Challenger.py* parsed the five separate response messages and reconstructed the fragmented digital signatures into a continuous digital signature pairing-object for digital signature verification. Figure 5.21 is a screenshot of *A_Challenger.py*'s digital signature re-assembly and verification status messages.

Phase 4 concluded as expected without any abnormalities or noteworthy events. Phase 4 goals were successfully achieved. Positive verification of the CAMINO-701's receive logs and positive observation of visual and auditory alarms validates Phase 4 success.

5.5 System Metrics

Section 5.5 discusses the recorded system metrics and juxtaposes the experimental AIS authenticator's performance against codified AIS data packet reporting and operational requirements in [7].

The ITU requires maritime vessels equipped with AIS to have regular data packet reporting intervals based on the vessel's dynamic conditions (e.g., velocity, course change) and equipped AIS transponder class. Ideally, the performance of the proposed AIS authenticator needs to either meet or exceed the ITU requirements in order to operate in a high traffic

```
**************************************************
AIVDM message received: !AIVDM,1,1,,A,<53BrgiGBUwPE4E>,0*5F

Message from 338999999
Message type: 12
**************************************************

##################################################
Initiating AIVDM message analysis...
##################################################

##################################################
Message received at/on: 11-Nov-2019 (16:09)
Message is from: 338999999
AIVDM Message Payload: <53BrgiGBUwPE4E>
Automatically initiating AIVDM message challenge processes...
##################################################

Calling createChallenge()...
##################################################
Initiating AIVDM to binary conversion process...
b'0<53BrgiGBUwPE4E>,0*5F'
AIVDM to binary conversion process successful...
Checking generated payload binary for 6-bit alignment...
Alignment complete...
Initiating conversion of challenge message payload to 6-bit ASCII...
Conversion to 6-bit ASCII complete...
Checking max 48 character payload size...
There are 1 total fragments...
----------------------------------------
Sending challenge message to: 338999999
Challenge Message: !AIABM,1,1,0,338999999,1,12,?3Dk@W9WJDM2EGM@ACA5?Rhh:SE6,0*69
Challenge Message Payload: ?3Dk@W9WJDM2EGM@ACA5?Rhh:SE6
Interrogated Message: <53BrgiGBUwPE4E>,0*5F

Transmitting challenge message to transponder...
Transponder has message for transmission...
Message Transmission Complete...

----------------------------------------

Challenge message creation & transmission complete...

##################################################
```

Figure 5.18. Various process states of Phase 2.

maritime environment. Due to time constraints, this research only measured end-to-end time and CPU-load percentages to determine the authenticator's suitability of practical operation in a traffic maritime environment and compliance with ITU standards.

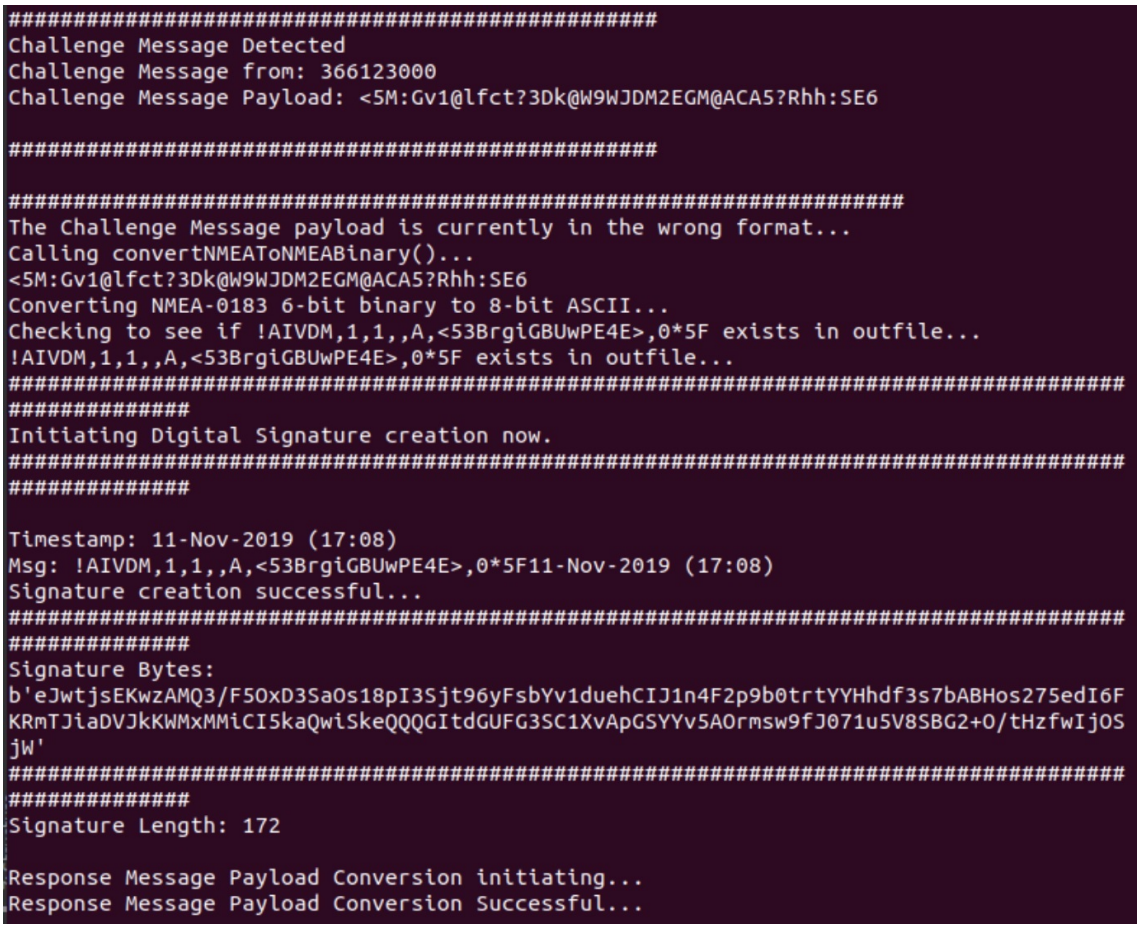


Figure 5.19. A display of B_Challenger's readout indicating the receipt of a challenge message from the CAMINO-701 transponder followed by the digital signature creation process.

5.5.1 Overall Successful Challenge Response Rate

Out of six trials (60 transmissions), the proposed system was operationally successful in 41.67 percent of the trials. Our research identified that the failures were caused by a problematic "checkfile" function. We identified the problematic function by referencing data strings in laptop buffers.

The FA-150 AIS authenticator script features a checkfile() function, which represents the experimental AIS authenticator's buffer used to store the initial, normal data packet transmission. During the experiments, the checkfile() function was often unreliable and could not locate an existing string value. This malfunction is the sole cause of failed test iterations

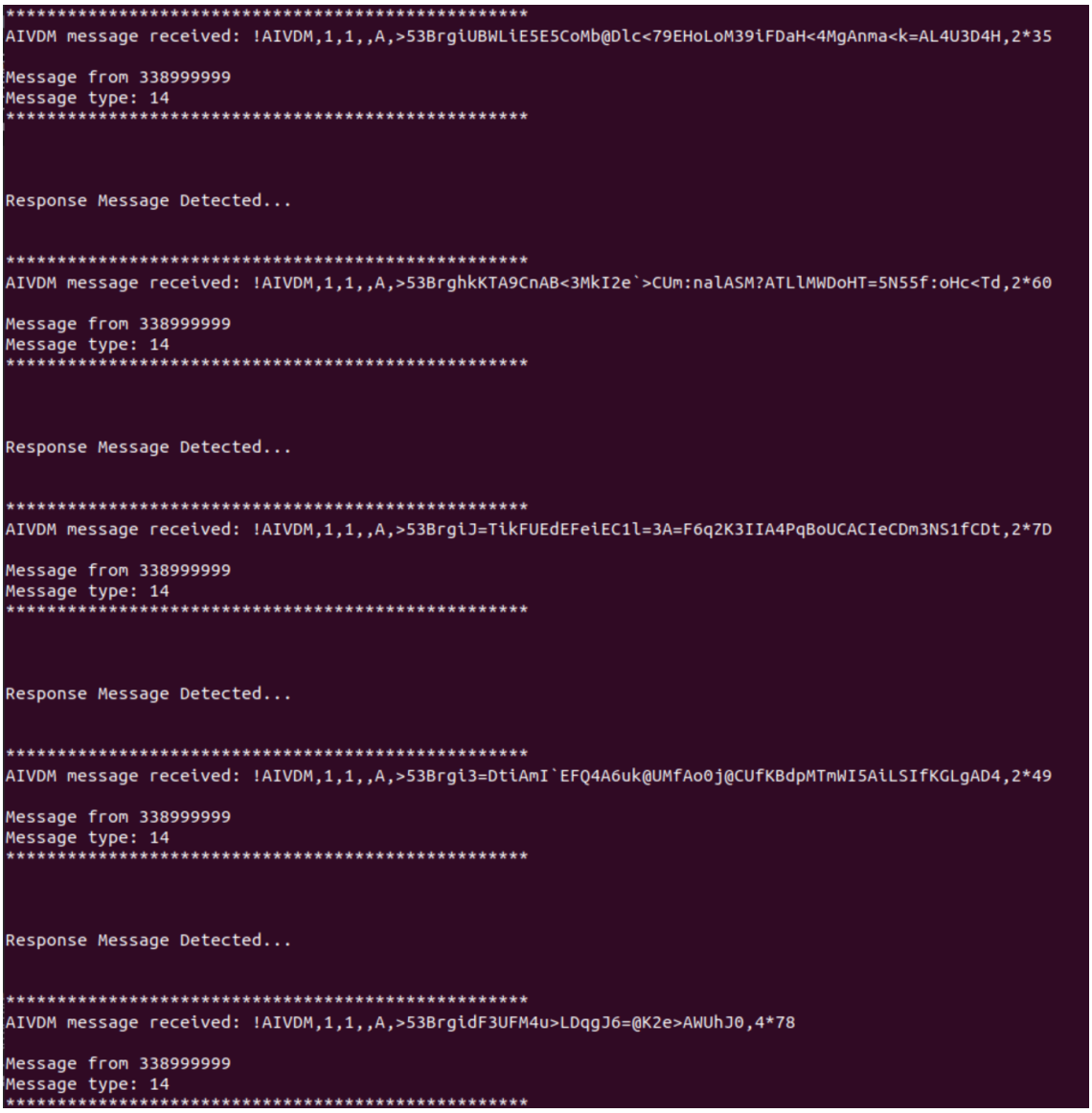


Figure 5.20. This figure depicts the five digital signature fragments originating from the FA-150.

during the experiments. The remaining portions of the authenticator functioned as designed. When the checkfile() function did not fail, the authenticator scripts created, fragmented, transmitted, reconstructed, and verified the digital signatures without errors or issues.

The validation process involved locating the data string contained in the challenge message with the contents of the outfile, which simulates the authenticator's internal buffer.

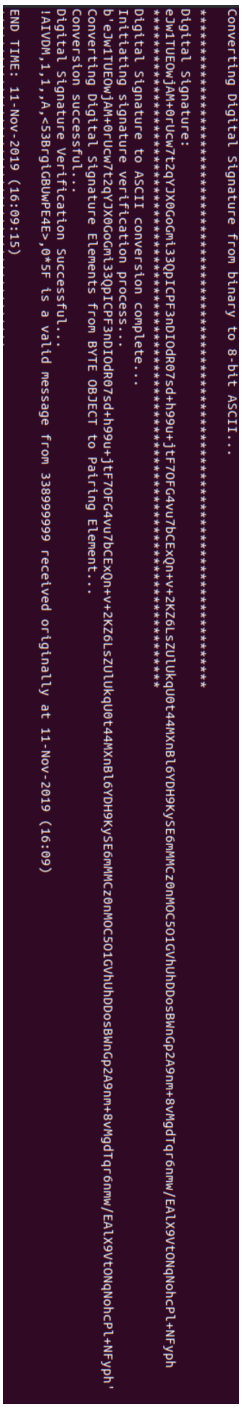


Figure 5.21. This figure depicts the successful reassembly and verification of the five digital signature fragments.

In all instances, regardless of success or failure, the designated outfile contained the targeted string value. We surmised that the 41.67 percent operational success rate may be improved through re-design and optimization of the checkfile() function.

5.5.2 System Response Time

The FA-150 was configured to transmit the ordinary data packets to the CAMINO-701 for challenge at one-minute intervals. A one-minute transmission interval was selected to provide sufficient time to observe and record both AIS authenticators' status messages and end-to-end times.

With regards to the transmission of the response messages, the FA-150's AIS authenticator was configured to transmit each digital signature fragment at one-second intervals. This transmission frequency was selected because it allowed for sufficient time to record data and observe various authenticator operational states. Furthermore, this transmission frequency produced reliable, positive results; whereas a higher frequency would cause data packet transmission irregularities.

Figure 5.22 depicts the data collected over six different test sessions. Each session included ten sets of data packet exchanges.

The resultant end-to-end times specify the time required to complete Phase 1 – 4 and provide an operational limitation of the proposed AIS authenticator prototype because this time describes how quickly the prototype is capable of processing one challenge-and-response data packet set before another arrives.

Table 5.1 summarizes the recorded metrics for system performance over six trial sets (60 transmissions):

Table 5.1. Summary of averaged recorded metrics for proposed system performance over 60 transmissions.

Summary of Recorded Metrics for System Performance over Six Trials Sets						
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6
Overall Success Rate	50%	10%	60%	50%	20%	60%
Average End-to-End Time	4.4 seconds	6 seconds	4.67 seconds	7 seconds	6 seconds	6 seconds

Trial 1

Time Initiated	Time Finished	Duration	String Does Not Exist?	Successful?
15:26:42	15:26:50	8 secs	N/A	Yes
15:27:42	15:27:46	4 secs	N/A	Yes
15:28:42	15:28:46	4 secs	N/A	Yes
15:29:42			Yes	No
15:30:42			Yes	No
15:31:42			Yes	No
15:32:42			Yes	No
15:33:42	15:33:47	3 secs	N/A	Yes
15:34:42			Yes	No
15:35:42	15:35:47	3 secs	Yes	Yes

Trial 2

Time Initiated	Time Finished	Duration	String Does Not Exist?	Successful?
15:37:42			Yes	No
15:38:42			Yes	No
15:39:42			Yes	No
15:40:42	15:40:47	6 sec	No	Yes
15:41:42			Yes	No
15:42:42			Yes	No
15:43:42			Yes	No
15:44:42			Yes	No
15:45:42			Yes	No
15:46:42			Yes	No

Trial 3

Time Initiated	Time Finished	Duration	String Does Not Exist?	Successful?
16:00:08	16:00:13	5 sec	N/A	Yes
16:01:08	16:01:13	5 sec	N/A	Yes
16:02:09	16:02:13	4 sec	N/A	Yes
16:03:09			Yes	No
16:04:09			Yes	No
16:05:09	16:05:13	4 sec	N/A	Yes
16:06:09			N/A	Yes
16:07:09	16:07:13	4 sec	N/A	Yes
16:08:09			Yes	No
16:09:09	16:09:15	6 sec	N/A	Yes

Trial 4

Time Initiated	Time Finished	Duration	String Does Not Exist?	Successful?
16:37:23	16:37:30	7 sec	N/A	Yes
16:38:23			Yes	No
16:39:23			Yes	No
16:40:23			Yes	No
16:41:23			Yes	No
16:42:23	16:42:30	7 sec	N/A	Yes
16:43:23	16:43:30	7 sec	N/A	Yes
16:44:23	16:44:30	7 sec	N/A	Yes
16:45:23			Yes	No
16:46:23	16:46:30	7 sec	N/A	Yes

Trial 5

Time Initiated	Time Finished	Duration	String Does Not Exist?	Successful?
16:47:23			Yes	No
16:48:24			Yes	No
16:49:24			Yes	No
16:50:24			Yes	No
16:51:24	16:51:30	6 sec	N/A	Yes
16:52:24	16:52:30	6 sec	N/A	Yes
16:53:24			Yes	No
16:54:24			Yes	No
16:55:24			Yes	No
16:56:24			Yes	No

Trial 6

Time Initiated	Time Finished	Duration	String Does Not Exist?	Successful?
16:59:34	16:59:40	6 sec	N/A	Yes
17:00:34	17:00:41	7 sec	N/A	Yes
17:01:34			Yes	No
17:02:34			Yes	No
17:03:34			Yes	No
17:04:35	17:04:41	6 sec	N/A	Yes
17:05:35	17:05:42	7 sec	N/A	Yes
17:06:35	17:06:42	7 sec	N/A	Yes
17:07:35			Yes	No
17:08:35	17:08:42	7 sec	N/A	Yes

Figure 5.22. AIS authenticator test data from 60 iterations.

Table 5.2 summarizes the overall testing results:

Table 5.2. Summary of overall system performance over six trial sets (60 transmissions).

Summary of Overall System Performance over Six Trials Sets	
Success Rate	41.67%
Average End-to-End Time	5.67 seconds
Lowest End-to-End Time	3 seconds
Highest End-to-End Time	8 seconds
Median End-to-End Time	6 seconds

For commercial implementation, the end-to-end time must be less than or equal to the AIS transponder's class-specific reporting interval because a slower time will not support the ITU's requirements for AIS transmission frequency. Figure 5.23 represents the Class A transponder's periodic reporting interval and Figure 5.24 represents the Class B transpon-

der's periodic reporting interval. The ITU specifies each transponder class's reporting interval on the vessel's dynamic data (e.g, heading change, velocity, moored, anchored).

Various factors such as organizational policy, cargo load, weight, fuel costs, route, and other global economic factors influence each vessel's cruising speed. Thus, the proposed AIS authenticator should support various contact densities with an end-to-end time that meets various Class A and B reporting intervals.

Agarwal [48] estimates the cruising speed of various Class A maritime vessels as follows:

- Bulk carriers – 13 to 15 knots
- Container ships – 16 to 24 knots
- Oil and chemical tankers – 13 to 17 knots
- "Roll on roll off" vessels – 16 to 22 knots
- Cruise ships – 20 to 25 knots

Class A shipborne mobile equipment reporting intervals[2]

Ship's dynamic conditions	Nominal reporting interval
Ship at anchor or moored and not moving faster than 3 knots	3 min(1)
Ship at anchor or moored and moving faster than 3 knots	10 s(1)
Ship 0-14 knots	10 s(1)
Ship 0-14 knots and changing course	3 1/3 s(1)
Ship 14-23 knots	6 s(1)
Ship 14-23 knots and changing course	2 s
Ship >23 knots	2 s
Ship >23 knots and changing course	2 s

(1) When a mobile station determines that it is the semaphore (see § 3.1.1.4, Annex 2), the reporting interval should decrease to 2 s (see § 3.1.3.3.2, Annex 2).

NOTE 1 – These values have been chosen to minimize unnecessary loading of the radio channels while maintaining compliance within the IMO AIS performance standards.

NOTE 2 – If the autonomous mode requires a shorter reporting interval than the assigned mode, the Class A shipborne mobile AIS station should use the autonomous mode.

Figure 5.23. Class A AIS shipboard mobile equipment reporting intervals based on various dynamic vessel states. Source: [7].

Given the cruising speed estimates, it is fair to assume that the majority of commercial vessels' transit speed ranges from 14 to 23 knots. Therefore, the majority of commercial

vessels transmit AIS data packets at a six-second interval (two seconds, if changing course), and other vessels transmit AIS data packets at 10 second intervals.

Based on the experimental results, shown in Figure 5.22, the AIS authenticator prototype is capable of reliably supporting vessels that are either stationary or at cruising speeds up to 14 knots. However, the prototype is currently not suitable for faster vessels whose transit requires faster end-to-end processing speeds.

Conversely, the authenticator prototype is capable of reliably supporting Class B-equipped vessels cruising up to 23 knots because the Class B reporting intervals are greater than the highest recorded end-to-end time.

Reporting intervals for equipment other than Class A shipborne mobile equipment[3]

Platform's condition	Nominal reporting interval	Increased reporting interval
Class B "SO" shipborne mobile equipment not moving faster than 2 knots	3 min	3 min
Class B "SO" shipborne mobile equipment moving 2–14 knots	30 s	30 s
Class B "SO" shipborne mobile equipment moving 14–23 knots	15 s	30 s(3)
Class B "SO" shipborne mobile equipment moving >23 knots	5 s	15 s(3)
Class B "CS" shipborne mobile equipment not moving faster than 2 knots	3 min	–
Class B "CS" shipborne mobile equipment moving faster than 2 knots	30 s	–
Search and rescue aircraft (airborne mobile equipment)	10 s(2)	–
Aids to navigation	3 min	–
AIS base station	10 s(1)	–

(1) The base station's reporting interval (RI) should decrease to 3 1/3 s after the station detects that one or more stations are synchronizing to the base station (see § 3.1.3.3.1, Annex 2).

(2) Shorter RI down to 2 s could be used in the area of search and rescue operations.

(3) Class B "SO" AIS shall report at the "Increased reporting interval" only when the last four consecutive frames each have less than 50% Free slots. Class B "SO" AIS shall not return to the "Normal reporting interval" until 65% or more of the slots of each of the last four consecutive frames are free.

Figure 5.24. Class B AIS shipboard mobile equipment reporting intervals based on various dynamic vessel states. Source: [7].

5.5.3 System CPU Load

Appendix K documents the CAMINO-701 authenticator's CPU load data and Appendix L documents the FA-150 authenticator's CPU load data. The research implemented a multi-threaded CPU load measuring function into the authenticator source codes for the CAMINO-701 and FA-150. Both authenticators' load data were captured utilizing the Python3 "psutil" library. The CPU load data was recorded at ten second intervals and it was automatically recorded into a designated text file. A ten-second interval was selected to provide sufficient sample points over a 30 minute test session.

The CPU load tests were conducted while utilizing two Virtual Machine (VM)s that were configured for one CPU core and 10 GB of virtual system memory. The VM supporting the CAMINO-701 authenticator possessed a CPU at 1.9GHz and the VM supporting the FA-150 authenticator possessed a CPU at 2.0GHz.

The research conducted a 30-minute stress test to measure three separate VM states: passive, idle, and active. Passive describes the idle state of the VM without the activated authenticator script. Idle describes the VM state when the authenticator script is actively listening for AIS data packets. Active describes the VM state when the authenticator script is fully listening, processing, and transmitting AIS data packets. Figure 5.25 and Figure 5.26 depict the CPU load metrics over 30 minutes for the CAMINO-701 authenticator and FA-150 authenticator, respectively. Noteworthy statistics for the CAMINO-701 authenticator's CPU load are given in Figure 5.27.

The CAMINO-701 authenticator's CPU load over a continuous 30 minute operational test resulted in an average percentage of 1.0 (passive), 0.6 (idle), and 3.1 (active); a minimum percentage of 0.0 (passive), 0.0 (idle), and 0.1 (active); and a maximum percentage of 6.1 (passive), 7.5 (idle), and 17.2 (active).

Noteworthy statistics for the FA-150 authenticator's CPU load are given in Figure 5.28. The FA-150 authenticator's CPU load over a continuous 30 minute operational test resulted in an average percentage of 1.9 (passive), 1.0 (idle), and 3.7 (active); a minimum percentage of 0.5 (passive), 0.2 (idle), and 0.8 (active); and a maximum percentage of 9.7 (passive), 6.1 (idle), and 24.1 (active).

Given the statistics, we conclude that the proposed authenticator prototype may function

adequately in low-density traffic environments because the simulated testing environment was equipped with two transponders programmed to communicate at one-minute intervals. Notwithstanding multiple outlying data points, both authenticators' active CPU load percentages consistently remained below 15 percent.

We expect that the CPU load for both authenticators may increase if the testing environment is reconfigured to resemble a high-density traffic environment. Currently, both authenticators' average and median active CPU load are between 2.4 percent and 3.7 percent per data packet processed. Additionally, the maximum CPU loads recorded were between 10 percent and 25 percent. From these experimental results, the authenticator prototype is not currently capable of supporting areas such the Strait of Dover, which is one of the many busy maritime routes in the world with a transit of approximately 400 vessels daily [28].

Although the research provided a proof-of-concept and added integrity and authenticity to AIS data packets without changing either the AIS standard or protocol, the proposed AIS authenticator is neither scalable nor capable of supporting high-density traffic environments under its current system specifications and un-optimized state. We expect that increasing the CPU cores, utilizing application-specific hardware instead of a general-use laptop VM, and optimizing the authenticator's code to increase efficiency and load-sharing capabilities will significantly improve performance times and scalability.

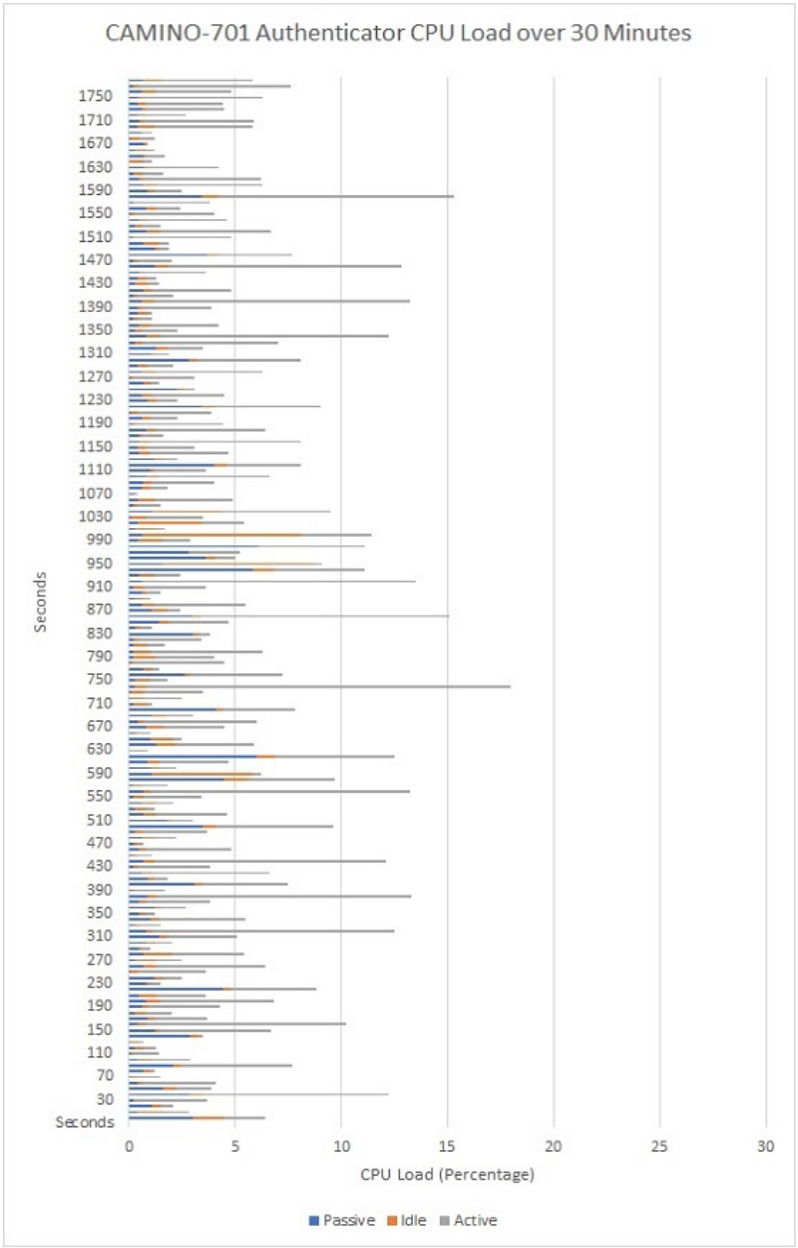


Figure 5.25. Recorded CPU load of the VM assigned to the CAMINO-701. CPU load was recorded under three conditions: passive, idle, and active.

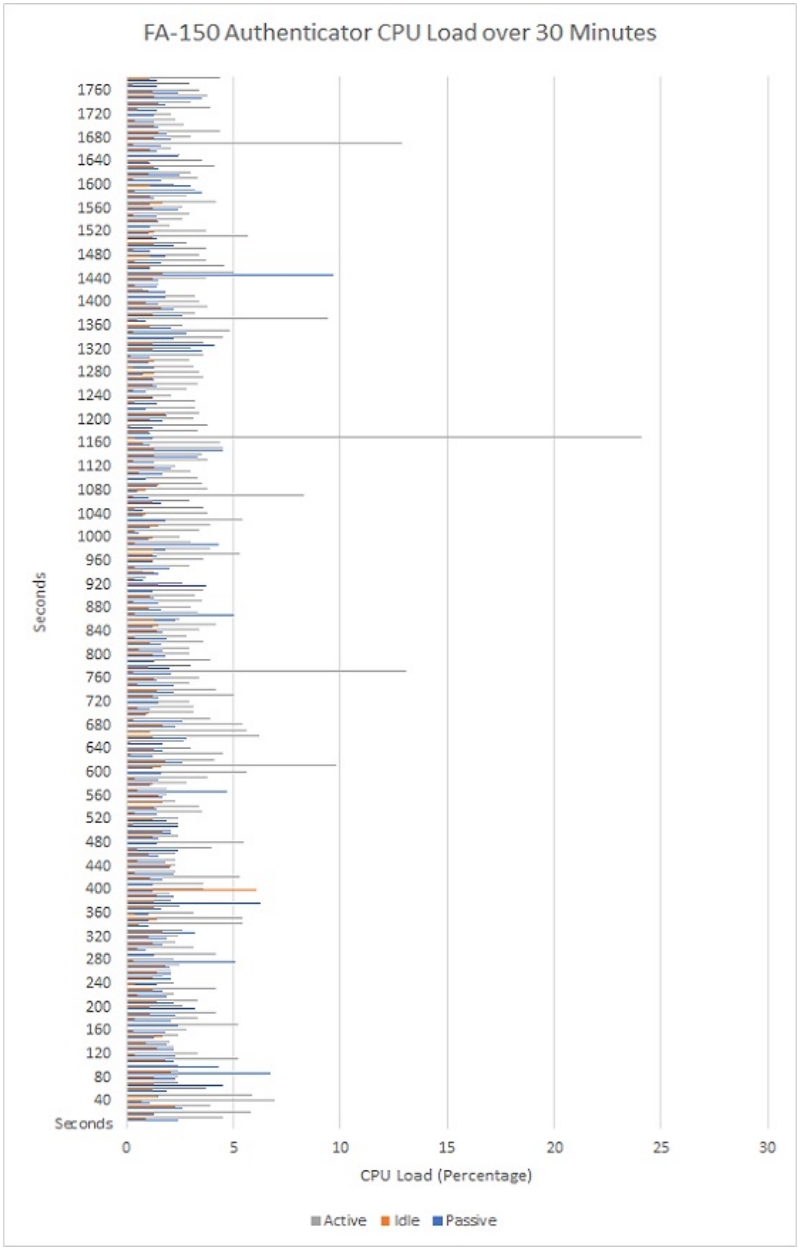


Figure 5.26. Recorded CPU load of the VM assigned to the FA-150. CPU load was recorded under three conditions: passive, idle, and active.

	Passive	Idle	Active
Average:	1.0	0.6	3.1
Mode:	0.2	0.4	0.4
Median:	0.6	0.4	2.7
Std:	1.1	0.9	3.0
Var:	1.3	0.8	9.0
Min:	0.0	0.0	0.1
Max:	6.1	7.5	17.2

Figure 5.27. Key statistics for recorded CPU load percentages of the VM assigned to the CAMINO-701. CPU load was recorded under three conditions: passive, idle, and active.

	Passive	Idle	Active
Average:	1.9	1.0	3.7
Mode:	1.4	1.2	2.4
Median:	1.6	1.1	3.3
Std:	1.1	0.6	2.3
Var:	1.3	0.4	5.1
Min:	0.5	0.2	0.8
Max:	9.7	6.1	24.1

Figure 5.28. Key statistics for recorded CPU load percentages of the VM assigned to the FA-150. CPU load was recorded under three conditions: passive, idle, and active.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Conclusions

This research has explored the addition of integrity and authentication capabilities to the AIS protocol without modification of the ITU-R M.1371-5 or NMEA-0183 standards. It showed how standard-compliant messages could be used to implement the protocol, examined possible cryptographic mechanisms for use in the protocol, and provided an initial proof-of-concept prototype.

The research prioritized and focused on validating system functionality through a proof-of-concept system using COTS AIS transmitters in the Engineering Enclave for Maritime Security (EEMS) Laboratory at the Naval Postgraduate School. The experimental results present performance metrics that may be improved upon through further development and optimization of the testbed components and the Python source codes.

The final authenticator design incorporated the use of ABM Input Messages to inject the challenge message and a BBM Input Message to inject the digital signature into the two AIS transponder test devices. The use of ABM and BBM resulted in a VDM Output Message that supports AIS RF transmissions from the transponder devices.

The research recorded an average experimental success rate of 41.67 percent and an average end-to-end time of 5.67 seconds over 60 test iterations. The overall test results demonstrate that the experimental AIS authenticator is compatible with the current AIS standard, functions as designed, and is capable of authenticating user-defined AIS data packets from current commercial AIS transponders.

Various mitigating actions were implemented to address a number of technical challenges (e.g., apparent manufacturer noncompliance with standards, AIS device design inadequacies, and AIS internal buffer limitations) that negatively affected the measured system performance (defined by success rate, end-to-end time, and CPU load percentage).

The collected data for average end-to-end time, success rate, and CPU-load indicates that the current design and implementation of the AIS authenticator is not currently capable of operating efficiently in a high-density traffic environment and therefore not supportive of

Class A AIS operating requirements. The CPU load on the prototype platform indicates that the AIS authenticator prototype may only be capable of reliable operation in low-density traffic environments without code or component optimization.

Our findings and recommendations are discussed in more detail in the following sections.

6.1 Research Questions

The following is a revisit of the three research questions introduced in Chapter 3 and a summary of the respective answers to the three questions.

What would the overall AIS technical architecture look like after the addition of message integrity and authentication? With a modified AIS authenticator design, AIS hardening features a three-mechanism data exchange between two entities to automatically challenge and respond appropriately against any AIS data packet via a cryptographic digital signature.

Does the current AIS protocol possess sufficient flexibility to provide message integrity and authentication without changing the overall architecture? The current AIS protocol and standard, outlined in ITU-R M.1371-5 and NMEA-0183, does provide sufficient flexibility to add integrity and authenticity to AIS data packets. A successful AIS authenticator prototype was engineered without any modifications to either AIS or NMEA standards.

To what extent does the current AIS protocol constrain the list of suitable cryptographic algorithms? The ABM's 48-character limit and BBM's 52-character limit for all ASM variant AIS Messages constrains the list of eligible cryptographic schemes for AIS authenticator implementation.

6.1.1 Design Constraints, Concerns, and Considerations

This research and the resultant AIS authenticator prototype is not currently scalable and capable of supporting Class A AIS vessels or operating in high-density traffic environments without code revision and optimization.

Given the CPU load measurements, this research concludes that an approximate 4 percent average CPU load, a 25 percent maximum CPU load, and consistent active CPU load

percentages between 5 and 15 percent are computationally intensive values given the singular ship-to-ship testing environment.

This research was able to engineer and implement an AIS authenticator prototype that complies with both the NMEA-0183 and ITU-R M.1371-5 standards. With regards to forward compatibility, the AIS authenticator is limited only by the current AIS protocol and standard. Despite the limitations of the AIS protocol and standard, the implemented authenticator is modular and possesses the flexibility to adapt over time to future changes to the AIS standard. Notwithstanding the technical issues during the research, the authenticator tested and operated well with legacy AIS devices as expected.

Although this research only tested and certified using one AIS message variant, the implemented AIS authenticator is capable of supporting the remaining 26 message types. This is possible because the authenticator's delivery methods and message encapsulation mechanisms function in a way that does not discriminate against different AIS message types.

The implemented AIS authenticator fulfilled the minimalist system constraint. The authenticator only utilized mechanisms and methods organic to the AIS protocol without the need for modifications.

6.2 Recommendations and Areas for Future Work

This section provides a discussion on the various recommendations and areas applicable for future work and further research. The following discussion will include recommendations and topics such as development of lightweight cryptographic algorithms, procurement and utilization of modern AIS equipment, simulation of real maritime traffic, utilization of application-specific integrated circuit boards, the optimization of *Challenger.py's* search function, and the implementation of a dynamic digital signature fragmentation function.

6.2.1 Identification and Development of Lightweight Cryptographic Algorithms

Future research on identifying suitable certificateless schemes or developing lightweight cryptographic algorithms may expand the list of AIS-compatible cryptosystems. The

AIS payload size, specifically, the ABM and BBM sentences constrain the list of eligible cryptographic schemes. The two standards limit the maximum possible digital signature size of any cryptographic algorithm or scheme to 1560 bits or 260 6-bit ASCII characters. As has been discussed in Chapter 3, certificateless-based schemes may be particularly relevant to the AIS environment, and identification and implementation of a relevant certificateless short signature scheme is a future research direction with excellent potential.

6.2.2 Procurement and Utilization of Modern AIS Equipment

Due to lengthy procurement and delivery times, this research only had access to two legacy AIS transponders models (i.e, FURNUO FA-150 and ALLTEK CAMINO-701). This was a research weakness because the newer FURUNO models, vice the CAMINO-701, would have provided support to both AIS message type 6 and 8. Therefore, it is recommended that future researchers have access to more than one late-model AIS transponder variant from any given manufacturer.

6.2.3 Simulate Real Maritime Traffic

NPS is located in Monterey, California and is in close proximity to the AIS traffic in Monterey Bay. However, this research did not make use of available, real maritime traffic. Instead, this research utilized simulated user-generated traffic from a pre-staged text file as an experimental control.

Even though both AIS transponders were within receiving range of Monterey Bay's AIS traffic, this research did not want to contaminate the maritime operating environment with spoofed AIS data packets. Each phase of the experiment generated volumes of experimental voyage data. This data, if leaked, would introduce irregularities, phantom ships, and hazards to real vessels and systems.

Therefore, a recommended improvement to this research is the implementation of maritime traffic that is more representative of a high-density maritime traffic area (e.g. Strait of Dover). Perhaps follow-on research could incorporate a maritime-traffic generator.

MarineTraffic (https://www.marinetraffic.com) is a ship tracking and maritime intelligence service provider that can supply future researchers with the necessary volumes of real

maritime traffic data. Additionally, MarineTraffic provides an application programming interface that allows developers and researchers to access and extract maritime data from a common repository.

6.2.4 Utilization of Application Specific Integrated Circuit Boards

In the beginning of Chapter 5, this thesis discusses the resultant CPU load percentages during active AIS message processing. The CPU load percentages measured as high as 25 percent and consistently between 5 and 15 percent. The prototype AIS authenticator's shared CPU and VM resources directly influenced these experimental results. Specifically, a single-core 2.0 GHz CPU on a Linux-based VM provided the processing power and resources to *A_Challenger.py* and a single-core 1.9 GHz CPU on a Linux-based VM provided the processing power and resources to *B_Challenger.py*.

This research recognizes that these system specifications and shared computational resources are not an optimal solution for the AIS authenticator. We recommend that future work and improvements to the AIS authenticator's hardware utilize application-specific integrated circuit boards and multi-core processors to reduce and load share CPU resources.

This research recommends that future researchers utilize the *Raspberry Pi 4* (RP4) as a possible low-cost hardware improvement if custom application-specific integrated circuit boards are not available. The RP4 is a flash-card sized integrated circuit board and light-weight computer solution. For example, Canakit (https://www.canakit.com/), an online electronics vendor, advertises a RP4 starter kit for approximately 120 US Dollars. This RP4 starter kit has the following pertinent system specifications: 1.5 GHz 64-bit quadcore ARMv8 CPU, 4GB RAM, and 32GB storage.

6.2.5 Optimization and Integration of AIS Authenticator Source Codes

This research utilized four distinct Python3 scripts (i.e., *KeyGenerationAuthority.py*, *A_Challenger.py*, *B_Challenger.py*, *InputEncoder.py*) to accomplish the specified research goals.

The AIS authenticator's code (i.e., *A_Challenger.py* and *B_Challenger.py*) prioritized func-

tionality rather than efficiency and optimization because this research was primarily focused on providing a proof-of-concept than a field-ready prototype.

KeyGenerationAuthority.py requires the least amount of future work due to its function of simulating the secure creation of a private key and the prior acquisition of a public key.

Similarly, *InputEncoder.py* does not require significant additional attention because its primary function was to simulate maritime traffic by generating custom user-defined ABM. Any improvement to *InputEncoder.py* should involve modifying the script to broadcast more frequently and provide scalable volumes of maritime traffic that better represents various maritime locales.

Future work should focus on the optimization of *A_Challenger.py*'s static, hard-coded digital signature queue and *B_Challenger.py*'s search function.

A_Challenger.py's digital signature fragment queue was designed as a static five-position, first-in-first-out queue that stored the digital signature fragments prior to reassembly and verification. We recommend further development and optimization of a dynamic digital signature queue and data reassembly mechanism to better accommodate future-developed, light-weight cryptographic algorithms that would require fewer consecutive messages than the maximum five time slots that the AIS protocol allows.

B_Challenger.py's search function was designed to take the challenge message's payload as an argument and then reference the designated outfile for the in-question data string's existence. The average experimental success rate was 41.67 percent because of the search function's inconsistencies. Even though the designated outfile always contained the in-question data string, the search function would occasionally fail to locate the targeted data string. We were unable to identify the specific cause of this failure. Consequently, this failure-to-locate generated a false-negative signal to follow-on functions and processes that triggered the AIS authenticator to ignore valid challenge messages.

Lastly, recall from Chapter 3 the proposal for Message 6 and Message 8 utilization and from Chapter 5 the reason for the deviation from the original proposal to Message 12 and Message 14. If future research has access to late-model AIS transponders, this research recommends the use of Message types 6 and 8 and integration of *A_Challenger.py* and *B_Challenger.py*

as a possible area for improvement of the AIS authenticator prototype. The integration of these two scripts would allow for true end-to-end testing and the prevention of mechanism confusion and message indiscernible (discussed in Section 5.4.2).

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A:
AIS Message Types and Description

Table A.1. AIS message types and their descriptions. Adapted from: [20].

AIS Message Types and Descriptions		
Message ID	Name	Description
1	Position report	Scheduled position report; Class A shipborne mobile equipment
2	Position report	Assigned scheduled position report; Class A shipborne mobile equipment
3	Position report	Special position report, response to interrogation; Class A shipborne mobile equipment
4	Base station report	Position, UTC, date and current slot number of base station
5	Static and voyage related data	Scheduled static and voyage related vessel data report, Class A shipborne mobile equipment
6	Binary addressed message	Binary data for addressed communication
7	Binary acknowledgement	Acknowledgement of received addressed binary data
8	Binary broadcast message	Binary data for broadcast communication

Table A.1 continued from previous page

AIS Message Types and Descriptions		
9	Standard SAR aircraft position report	Position report for airborne stations involved in SAR operations only
10	UTC/date inquiry	Request UTC and date
11	UTC/date response	Current UTC and date if available
12	Addressed safety related message	Safety related data for addressed communication
13	Safety related acknowledgement	Acknowledgement of received addressed safety related message
14	Safety related broadcast message	Safety related data for broadcast communication
15	Interrogation	Request for a specific message type can result in multiple responses from one or several stations
16	Assignment mode command	Assignment of a specific report behavior by competent authority using a Base station
17	DGNSS broadcast binary message	DGNSS corrections provided by a base station
18	Standard Class B equipment position report	Standard position report for Class B shipborne mobile equipment to be used instead of Messages 1, 2, 3
19	Extended Class B equipment position report	No longer required. Extended position report for Class B shipborne mobile equipment; contains additional static information

Table A.1 continued from previous page

AIS Message Types and Descriptions		
20	Data link management message	Reserve slots for Base station(s)
21	Aids-to-navigation report	Position and status report for aids-to-navigation
22	Channel management	Management of channels and transceiver modes by a Base station
23	Group assignment command	Assignment of a specific report behavior by competent authority using a Base station to a specific group of mobiles
24	Static data report	Additional data assigned to an MMSI Part A: Name Part B: Static Data
25	Single slot binary message	Short unscheduled binary data transmission Broadcast or addressed
26	Multiple slot binary message with Communications State	Scheduled binary data transmission Broadcast or addressed
27	Position report for long range applications	Class A and Class B “SO” shipborne mobile equipment outside base station coverage

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: Alphabet_Soup.PDF

There are four parts to Appendix B.

B.1 Overall AIS to NMEA Conversion Process

This first of four figures details a high level diagram that describes the overall AIS and NMEA character conversion process of AIS Message 12 and Message 14. This first part also details the breakout of a NMEA ABM and BBM type Input Messages; this breakout includes where with the NMEA sentence the sample text, "THIS IS A TEST MESSAGE", is located and how it appears after 6-bit ASCII conversation.

B.2 Detailed Breakout of the AIS to NMEA Conversion Process

This second of four figures details the data conversion process identified in part 1 of Appendix B. This figure represents the normal process that a user performs via the AIS user-system console to create user-defined messages (e.g., a bridge operator inputs "Hello, World" to broadcast to another ship). The AIS protocol utilizes a specialized 6-bit ASCII alphabet while the NMEA protocol utilizes a different 6-bit ASCII alphabet.

In this example, "THIS IS A TEST MESSAGE" serves as the sample message. The first necessary step is to identify the 6-bit binary representation of each character in the sample message. The first character, "T", is equivalent to "010100" in the AIS alphabet. The second step is to identify the NMEA 6-bit ASCII representation of "010100"; "D" represents "010100".

Repeating the 2-step translation process for the remaining characters (including spaces) in the sample text yields: "D89CP9CP1PD5CDP".

B.3 Conversion of Digital Signatures to NMEA 6-bit ASCII

This third of four figures details the digital signature to NMEA sentence format. This conversion process is a slight modification of the process detailed in the Section B2 of this appendix.

The proposed authentication system assumes each digital signature takes an 8-bit ASCII alphanumeric format. The first step of the digital signature to NMEA conversion process is to convert each 8-bit ASCII in the digital signature to its binary representation. The second step is to translate each 6-bit binary set to its appropriate NMEA 6-bit ASCII representation. If the digital signature binary data is not a perfect factor of six, filler bits ("0") are used.

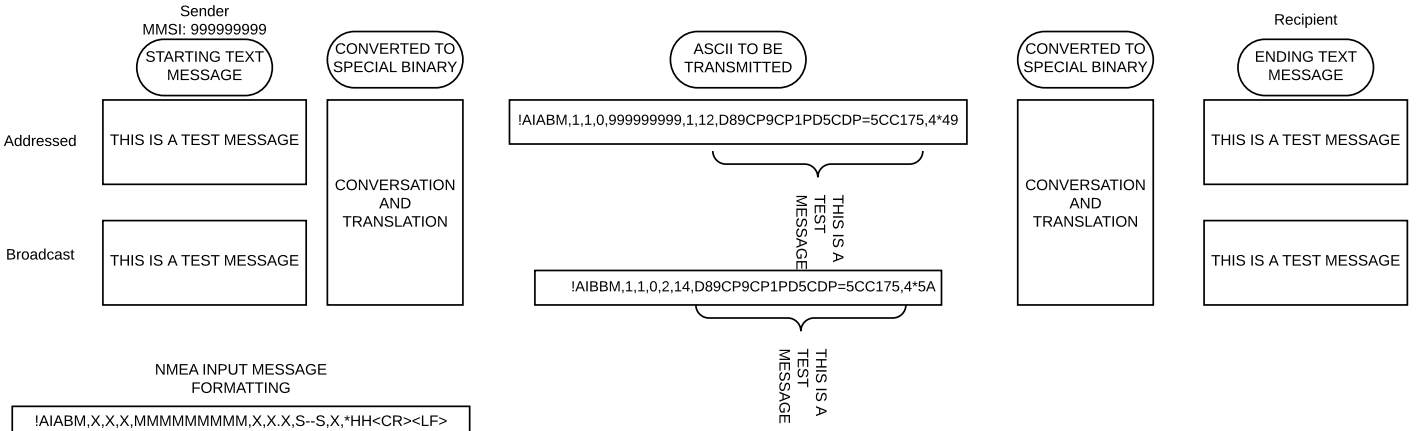
The 8-bit ASCII sample text, "0123456789:;<=" (requiring four filler bits) translates to "klu6WEcnK@".

B.4 Detailed Breakout of DAC and FI Codes

This fourth figure details how DAC and FI codes are applied to the conversion process and are embedded into the NMEA sentence.

The ITU-R M.1371-5 and NMEA-0183 do not explicitly mention how the DAC and FI codes are converted and embedded into the NMEA sentence structure for transmission via ABM or BBM Input Messages. This figure represents the results of the reverse engineering efforts to uncover the technical implementations of the DAC and FI codes.

Overall AIS/NMEA Character Conversion Process
For Message 12 & 14



NMEA INPUT MESSAGE FORMATTING

!AIABM,X,X,X,MMMMMMMMM,X,X.X,S--S,X,*HH<CR><LF>

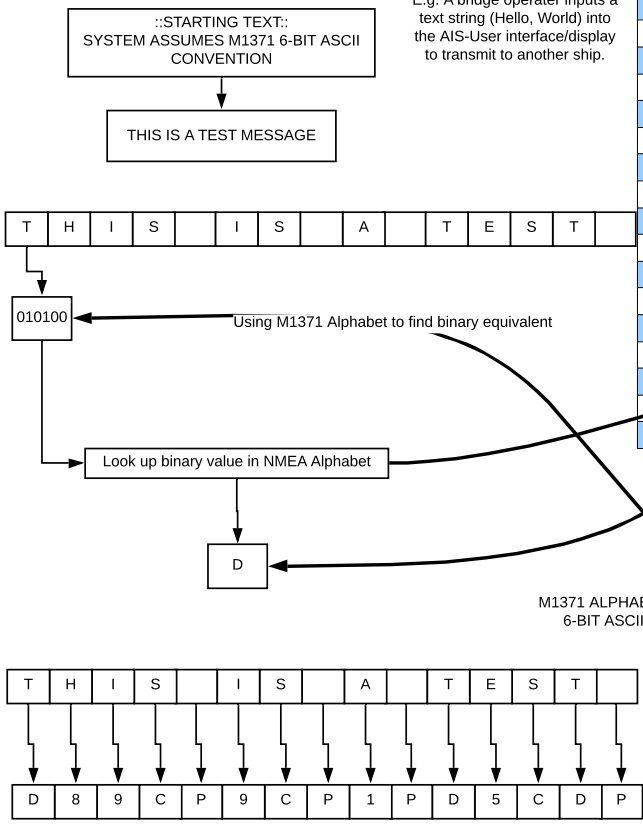
Checksum
Number of fill bits; Value: 0 - 5
Encapsulated data; 1 - 56 Bytes
VHF Datalink message number; Value: 12
AIS channel for transmission of radio message; Value: 0-No Preference, 1: A, 2: B, or 3-Both
MMSI of destnation AIS unit
Sequential message identifer; Value: 0-3
Sequence number; Value: 1 to 9
Total number of sentences needed to transfer the message; Value: 1 to 9

!AIBBM,X,X,X,X,X.X,S--S,X,*HH<CR><LF>

Checksum
Number of fill bits; Value: 0 - 5
Encapsulated data; 1 - 57 Bytes
VHF Datalink message number; Value: 12
AIS channel for transmission of radio message; Value: 0-No Preference, 1: A, 2: B, or 3-Both
Sequential message identifer; Value: 0-3
Sequence number; Value: 1 to 9
Total number of sentences needed to transfer the message; Value: 1 to 9

CHARACTER CONVERSION PROCESS*

*This is the normal process utilized by the AIS system to craft user-defined messages. E.g. A bridge operater inputs a text string (Hello, World) into the AIS-User interface/display to transmit to another ship.



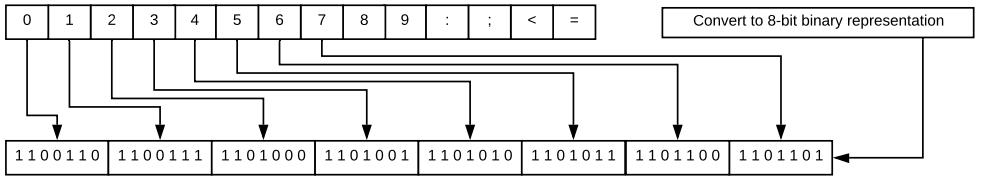
NMEA ALPHABET
6-BIT ASCII

Char	ASCII	Decimal	Binary	Char (1)	ASCII (1)	Decimal (1)	Binary (1)	Char (2)	ASCII (2)	Decimal (2)	Binary (2)	Char (3)	ASCII (3)	Decimal (3)	Binary (3)
"0"	48	0	0	"@"	64	16	10000	"P"	80	32	100000	"h"	104	48	110000
"1"	49	1	1	"A"	65	17	10001	"Q"	81	33	100001	"i"	105	49	110001
"2"	50	2	10	"B"	66	18	10010	"R"	82	34	100010	"j"	106	50	110010
"3"	51	3	11	"C"	67	19	10011	"S"	83	35	100011	"k"	107	51	110011
"4"	52	4	100	"D"	68	20	10100	"T"	84	36	100100	"l"	108	52	110100
"5"	53	5	101	"E"	69	21	10101	"U"	85	37	100101	"m"	109	53	110101
"6"	54	6	110	"F"	70	22	10110	"V"	86	38	100110	"n"	110	54	110110
"7"	55	7	111	"G"	71	23	10111	"W"	87	39	100111	"o"	111	55	110111
"8"	56	8	1000	"H"	72	24	11000	"X"	88	40	101000	"p"	112	56	111000
"9"	57	9	1001	"I"	73	25	11001	"a"	97	41	101001	"q"	113	57	111001
":"	58	10	1010	"J"	74	26	11010	"b"	98	42	101010	"r"	114	58	111010
";"	59	11	1011	"K"	75	27	11011	"c"	99	43	101011	"s"	115	59	111011
"<"	60	12	1100	"L"	76	28	11100	"d"	100	44	101100	"t"	116	60	111100
"="	61	13	1101	"M"	77	29	11101	"e"	101	45	101101	"u"	117	61	111101
">"	62	14	1110	"N"	78	30	11110	"f"	102	46	101110	"v"	118	62	111110
"?"	63	15	1111	"O"	79	31	11111	"g"	103	47	101111	"w"	119	63	111111

BINARY	DECIMAL	CHAR	BINARY (1)	DECIMAL (1)	CHAR (1)	BINARY (2)	DECIMAL (2)	CHAR (2)	BINARY (3)	DECIMAL (3)	CHAR (3)
0	0	"@"	10000	16	"P"	100000	32	" "	110000	48	"0"
1	1	"A"	10001	17	"Q"	100001	33	"!"	110001	49	"1"
10	2	"B"	10010	18	"R"	100010	34	"""	110010	50	"2"
11	3	"C"	10011	19	"S"	100011	35	"\#"	110011	51	"3"
100	4	"D"	10100	20	"T"	100100	36	"$"	110100	52	"4"
101	5	"E"	10101	21	"U"	100101	37	"%"	110101	53	"5"
110	6	"F"	10110	22	"V"	100110	38	"&"	110110	54	"6"
111	7	"G"	10111	23	"W"	100111	39	"\'"	110111	55	"7"
1000	8	"H"	11000	24	"X"	101000	40	"("	111000	56	"8"
1001	9	"I"	11001	25	"Y"	101001	41	")"	111000	56	"9"
1010	10	"J"	11010	26	"Z"	101010	42	"*"	111010	58	":"
1011	11	"K"	11011	27	"["	101011	43	"\+"	111011	59	";"
1100	12	"L"	11100	28	"\"	101100	44	","	111100	60	"<"
1101	13	"M"	11101	29	"]"	101101	45	"-"	111101	61	"="
1110	14	"N"	11110	30	"\^"	101110	46	"."	111110	62	">"
1111	15	"O"	11111	31	"_"	101111	47	"/"	111111	63	"?"

Modified Conversion Method to Transmit Digital Signatures based in 8-bit ASCII

::START::
DIGITAL SIGNATURE EXAMPLE



Partial 8-bit ASCII Representation

Hex	Char	Binary	Hex (1)	Char (1)	Binary (1)
A	<LF>	1010	64	.	1100100
D	<CR>	1101	65	\/	1100101
50	SPACE	1010000	66	0	1100110
51	!	1010001	67	1	1100111
52	"	1010010	68	2	1101000
53	#	1010011	69	3	1101001
54	$	1010100	70	4	1110000
55	%	1010101	71	5	1110001
56	&	1010110	72	6	1110010
57	\`	1010111	73	7	1110011
58	(	1011000	74	8	1110100
59	)	1011001	75	9	1110101
60	*	1100000	76	:	1110110
61	+	1100001	77	;	1110111
62	\'	1100010	78	<	1111000
63	\-	1100011	79	=	1111001

NMEA ALPHABET
6-BIT ASCII

Char	ASCII	Decimal	Binary	Char (1)	ASCII (1)	Decimal (1)	Binary (1)	Char (2)	ASCII (2)	Decimal (2)	Binary (2)	Char (3)	ASCII (3)	Decimal (3)	Binary (3)
"0"	48	0	0	"@"	64	16	10000	"P"	80	32	100000	"h"	104	48	110000
"1"	49	1	1	"A"	65	17	10001	"Q"	81	33	100001	"i"	105	49	110001
"2"	50	2	10	"B"	66	18	10010	"R"	82	34	100010	"j"	106	50	110010
"3"	51	3	11	"C"	67	19	10011	"S"	83	35	100011	"k"	107	51	110011
"4"	52	4	100	"D"	68	20	10100	"T"	84	36	100100	"l"	108	52	110100
"5"	53	5	101	"E"	69	21	10101	"U"	85	37	100101	"m"	109	53	110101
"6"	54	6	110	"F"	70	22	10110	"V"	86	38	100110	"n"	110	54	110110
"7"	55	7	111	"G"	71	23	10111	"W"	87	39	100111	"o"	111	55	110111
"8"	56	8	1000	"H"	72	24	11000	"X"	88	40	101000	"p"	112	56	111000
"9"	57	9	1001	"I"	73	25	11001	"a"	97	41	101001	"q"	113	57	111001
":"	58	10	1010	"J"	74	26	11010	"b"	98	42	101010	"r"	114	58	111010
";"	59	11	1011	"K"	75	27	11011	"c"	99	43	101011	"s"	115	59	111011
"<"	60	12	1100	"L"	76	28	11100	"d"	100	44	101100	"t"	116	60	111100
"="	61	13	1101	"M"	77	29	11101	"e"	101	45	101101	"u"	117	61	111101
">"	62	14	1110	"N"	78	30	11110	"f"	102	46	101110	"v"	118	62	111110
"?"	63	15	1111	"O"	79	31	11111	"g"	103	47	101111	"w"	119	63	111111

HOW DESIGNATED AREA CODES (DAC) AND FUNCTION IDENTIFIERS (FI) ARE APPLIED TO CONVERSION PROCESS AND EMBEDDED INTO THE NMEA PAYLOAD

DACs and FIs are unique identifiers required for Application-Specific Message (ASM). Without these two codes, the received AIS payload would not properly route to the appropriate maritime subsystem/application.

The literature (M1371-5 & NMEA-0183) does not explicitly mention how the two codes are converted and embedded into the NMEA-0183 sentence structure for transmission via !AIABM or !AIBBM input messages.

The following is a reverse engineering process utilized to locate the DAC and FI.

*Both sample sentences were constructed utilizing the automated ABM/BBM sentence encoder available at **https://www.maritec.co.za/aisabmbbmencode.php**

However, the encoder indicates an erroneous value for fill bits utilized. The encoder states "6". This not possible because the only values allowed are 0 through 5. The sample sentences does not reflect the correct amount of fill bits because it is irrelevant to the analysis.

!AIABM Input Sentence for Message 6: Addrssed Binary Message

Parameters:
Recipient MMSI: 999999999
Broadcast Channel: Both
Message: HELLO
DAC: 111
FI: 11

!AIABM,1,1,0,999999999,3,6,6tdPDhht,6*3A

6tdPDhht is currently in NMEA 6-bit ASCII format

Find the 6-bit binary equivalent

6	t	d	P	D	h	h	t
0 0 0 1 1 0	1 1 1 1 0 0	1 0 1 1 0 0	1 0 0 0 0 0	0 1 0 1 0 0	1 1 0 0 0 0	1 1 0 0 0 0	1 1 1 1 0 0

DAC	FI	H	E	L	L	O	
0 0 0 1 1 0 1 1 1 1	0 0 1 0 1 1	0 0 1 0 0 0	0 0 0 1 0 1	0 0 1 1 0 0	0 0 1 1 0 0	0 0 1 1 1 1	0 0

2 fill bits added for 6-bit alignment

Binary Values:
DAC = 0 0 0 1 1 0 1 1 1 1
FI = 0 0 1 0 1 1

M1371 Binary Value:
H = 0 0 1 0 0 0
E = 0 0 0 1 0 1
L = 0 0 1 1 0 0
O = 0 0 1 1 1 1

!AIBBM Input Sentence for Message 8: Broadcast Binary Message

Parameters:
Recipient MMSI: 999999999
Broadcast Channel: Both
Message: HELLO
DAC: 1
FI: 10

!AIBBM,1,1,0,0,8,04`PDhht,6*24

04`PDhht is currently in NMEA 6-bit ASCII format

Find the 6-bit binary equivalent

0	4	`	P	D	h	h	t
0 0 0 0 0 0	0 0 0 1 0 0	1 0 1 0 0 0	1 0 0 0 0 0	0 1 0 1 0 0	1 1 0 0 0 0	1 1 0 0 0 0	1 1 1 1 0 0

DAC	FI	H	E	L	L	O	
0 0 0 0 0 0 0 0 0 1	0 0 1 0 1 0	0 0 1 0 0 0	0 0 0 1 0 1	0 0 1 1 0 0	0 0 1 1 0 0	0 0 1 1 1 1	0 0

2 fill bits added for 6-bit alignment

Binary Values:
DAC = 0 0 0 0 0 0 0 0 0 1
FI = 0 0 1 0 1 0

M1371 Binary Value:
H = 0 0 1 0 0 0
E = 0 0 0 1 0 1
L = 0 0 1 1 0 0
O = 0 0 1 1 1 1

APPENDIX C:
KeyGenerationAuthority.PY

The majority of the Python source code found in this appendix is from the Charm Cryptography Library and utilizes the Stanford PBC Library.

This script was designed to simulate a key generation authority and generate the KGA's master public key and each of the two ships' private keys.

Our modification to this code is found under the main() function. Specifically, we added the nine lines of code that interacted with "privatekey.txt", "privatekey1.txt", and "privatekey2.txt".

The additional lines of code located under the execution of main() were "commented out" and used as reference during the research.

```
'''

The following code is from the Charm Cryptography Python
Module, which utilizes the Stanford PBC Library.

KeyGenerationAuthority.py was designed to simulate a Key
Generation Authority. This script will generate and write
into file the KGC's master public key and each of the two
ship's private keys. The utilization of a .txt file
simulates the secure transmission and receipt of the
private key from KGC to each unique entity.

:Boneh-Lynn-Shacham Identity Based Signature

| From: "D. Boneh, B. Lynn, H. Shacham Short Signatures
| from the Weil Pairing"
| Published in: Journal of Cryptology 2004
| Available from: http://
```

```
| Notes: This is the IBE (2-level HIBE) implementation of
    the HIBE scheme BB_2.

* type:           signature (identity-based)
* setting:        bilinear groups (asymmetric)

:Authors:    J. Ayo Akinyele
:Date:       1/2011
'''

from charm.toolbox.pairinggroup import PairingGroup, ZR, G1,
    G2, pair
from charm.core.engine.util import objectToBytes
from charm.toolbox.IBSig import *


class BLS01(IBSig):

    def __init__(self, groupObj):
        IBSig.__init__(self)
        global group
        group = groupObj


    def dump(self, obj):
        return objectToBytes(obj, group)


    def keygen(self, secparam=None):
        g, x = group.random(G2), group.random()
        g_x = g ** x
        pk = {'g^x': g_x, 'g': g, 'identity': str(g_x), '
            secparam': secparam}
```

```
        sk = {'x': x}
        return (pk, sk)


def main():
    groupObj = PairingGroup('MNT159')
    bls = BLS01(groupObj)

    (pk, sk) = bls.keygen()

    x = objectToBytes(sk['x'], group)
    g = objectToBytes(pk['g'], group)
    gx = objectToBytes(pk['g^x'],group)

    with open("privatekey.txt", "wb+") as sk:
        sk.write(x)
        sk.close()

    with open("publickey1.txt", "wb+") as pk1:
        pk1.write(g)
        pk1.close()

    with open("publickey2.txt", "wb+") as pk2:
        pk2.write(gx)
        pk2.close()



if __name__ == "__main__":
    debug = True
    main()
```

```
'''

Waters - Identity-based signatures

| From: "B. Waters - Efficient identity-based encryption
   without random oracles"
| Published in: EUROCRYPT 2005
| Available from: Vol 3494 of LNCS, pages 320-329
| Notes:

* type:           signature (ID-based)
* setting:        bilinear groups (asymmetric)

:Authors:    J. Ayo Akinyele
:Date:       11/2011


from charm.toolbox.pairinggroup import PairingGroup,ZR,G1,G2
   ,pair
from charm.toolbox.iterate import dotprod
from charm.toolbox.hash_module import Waters
from charm.core.engine.util import *


debug = False

class WatersSig:

    def __init__(self, groupObj):
        global group,lam_func
        group = groupObj
```

```
    lam_func = lambda i,a,b: a[i] ** b[i]


def setup(self, z, l=32):
    global waters

    waters = Waters(group, z, l)
    alpha, h = group.random(ZR), group.random(G1)
    g1, g2 = group.random(G1), group.random(G2)
    A = pair(h, g2) ** alpha
    y = [group.random(ZR) for i in range(z)]
    y1t,y2t = group.random(ZR), group.random(ZR)
    u1t = g1 ** y1t; u2t = g1 ** y2t
    u = [g1 ** y[i] for i in range(z)]
    u1b = g2 ** y1t; u2b = g2 ** y2t
    ub =[g2 ** y[i] for i in range(z)]
    msk = h ** alpha
    mpk = {'g1':g1, 'g2':g2, 'A':A, 'u1t':u1t, 'u2t':u2t
       , 'u':u, 'u1b':u1b, 'u2b':u2b, 'ub':ub, 'z':z, 'l
       ':l }

    return (mpk, msk)


def keygen(self, mpk, msk, ID):
    if debug: print("Keygen alg...")

    k = waters.hash(ID) # return list from k1,...,kz

    if debug: print("k =>", k)

    r = group.random(ZR)
    k1 = msk * ((mpk['u1t'] * dotprod(1, -1, mpk['z'],
```

```
        lam_func, mpk['u'], k)) ** r)
    k2 = mpk['g1'] ** -r

    return (k1, k2)


def sign(self, mpk, sk, M):
    if debug: print("Sign alg...")

    m = waters.hash(M) # return list from m1,...,mz

    if debug: print("m =>", m)

    (k1, k2) = sk
    s  = group.random(ZR)
    S1 = k1 * ((mpk['u2t'] * dotprod(1, -1, mpk['z'],
        lam_func, mpk['u'], m)) ** s)
    S2 = k2
    S3 = mpk['g1'] ** -s

    return {'S1':S1, 'S2':S2, 'S3':S3}



def verify(self, mpk, ID, M, sig):
    if debug: print("Verify...")

    k = waters.hash(ID)
    m = waters.hash(M)
    (S1, S2, S3) = sig['S1'], sig['S2'], sig['S3']
    A, g2 = mpk['A'], mpk['g2']
    comp1 = dotprod(1, -1, mpk['z'], lam_func, mpk['ub
        '], k)
```

```
        comp2 = dotprod(1, -1, mpk['z'], lam_func, mpk['ub
            '], m)
        lhs = (pair(S1, g2) * pair(S2, mpk['u1b'] * comp1) *
             pair(S3, mpk['u2b'] * comp2))
        #if ((pair(S1, g2) * pair(S2, mpk['u1b'] * comp1) *
            pair(S3, mpk['u2b'] * comp2)) == A):

        if lhs == A:
            return True

        return False


def main():

    groupObj = PairingGroup('MNT159')
    wat = WatersSig(groupObj)
    (master_public_key, master_secret_key) = wat.setup(5)

    MMSI1 = "366123000"  #These values are hard coded into
        this script to accommodate
    MMSI2 = "338999999"  #for the two AIS transponder units.


    #Generation of unique private keys for each AIS
        transponder units.

    one_secret_key = wat.keygen(master_public_key,
        master_secret_key, MMSI1)
    two_secret_key = wat.keygen(master_public_key,
        master_secret_key, MMSI2)

    #Conversion of unique private keys into byte objects in
```

```
        order to write to file.
    one_secret_key_bytes = objectToBytes(one_secret_key,
        group)
    two_secret_key_bytes = objectToBytes(two_secret_key,
        group)

    #Conversion of KGC's master public key into byte object
        in order to write to file.
    mpk_bytes = objectToBytes(master_public_key, group)

    with open("ShipONE.txt", "wb+") as w1:
        w1.write(one_secret_key_bytes)
        w1.close()

    with open("ShipTWO.txt", "wb+") as w2:
        w2.write(two_secret_key_bytes)
        w2.close()

    with open("masterpublic.txt", "wb+") as wmpk:
        wmpk.write(mpk_bytes)
        wmpk.close()

if __name__ == "__main__":
    debug = False
    main()

'''
```

APPENDIX D: *KeyGenerationAuthority.PY* Process Flow Graph

The figure located in this appendix details the overall process flow for *KeyGenerationAuthority.py*. The cryptographic portions of this source code is found within the Charm Cryptography Library.

The Charm Cryptography module generates cryptographic values as pairing objects and dictionary objects. However, in order to write the digital cryptographic keys to a text file the information requires conversion to byteobject format.

All technical specifics of the module is found within Charm Cryptogrpahy Library's documentation.

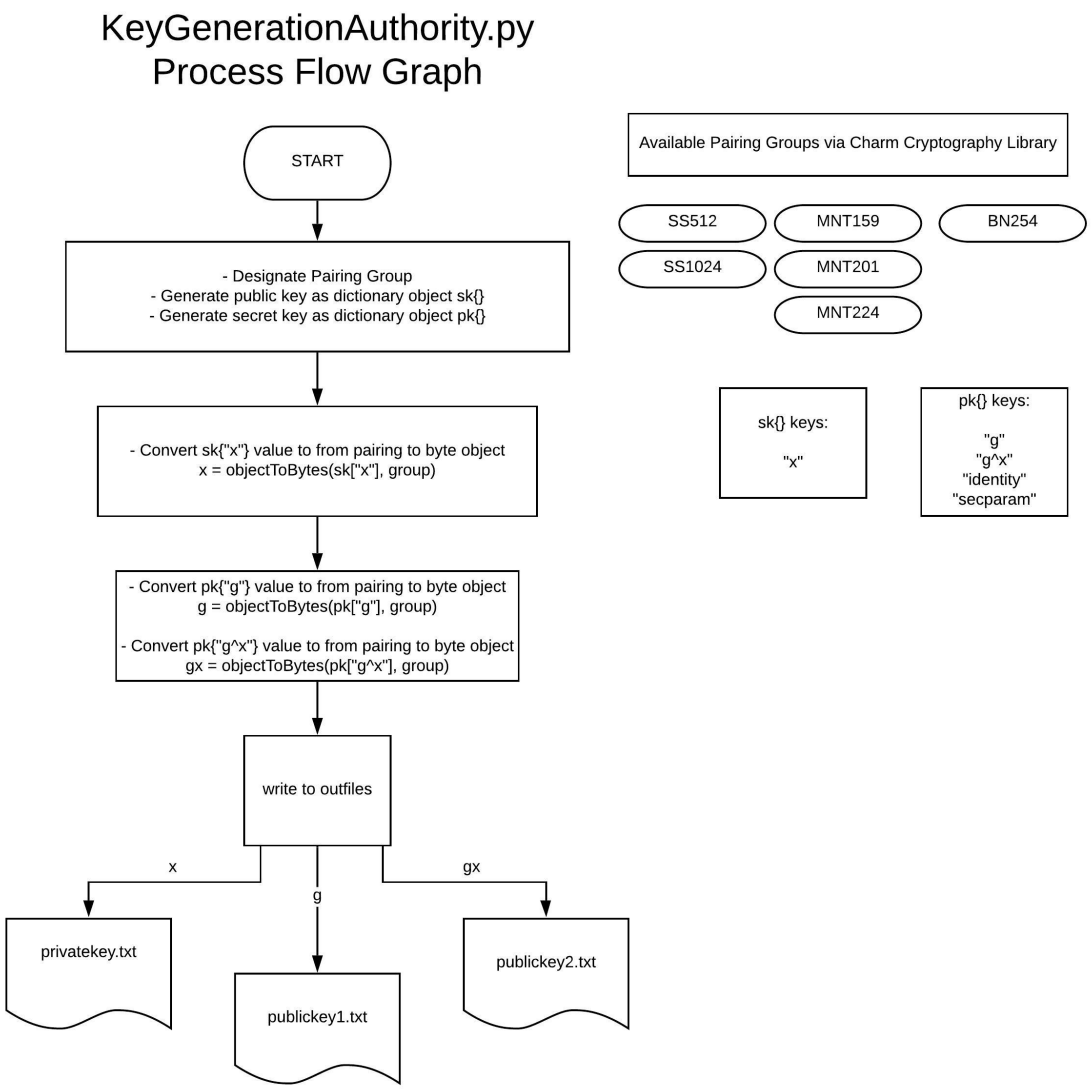


Figure D.1. *KeyGenerationAuthority.py* Process Flow Graph

APPENDIX E:
InputEncoder.PY

```
# This scripts takes an input text file and creates a 6bit
# ASCII equivalent.

# ABM - Addressed binary and safety related message
# Used to input NMEA format messages into AIS transponder
# for output as RF.
# For AIS Message Types 6, 12, 25, 26
#
# !AIABM,x,x,x,xxxxxxxxx,x,x.x,s--s,x,*hh<CR><LF>
#        | | |     |     | | | |   |
#        | | |     |     | | |   |    `-Number of fill bits
   (0-5)
#        | | |     |     | | |    `--Encapsulated data
#        | | |     |     | |  `--VHF datalink message number
   (6, 12, 25, 26), see ITU-R M1371
#        | | |     |      `--AIS channel for broadcast of
   the radio message (0-3)
#        | | |     `--The MMSI of destination AIS unit for
   the ITU-R M1371 message (9 digits; 30 bits)
#        | | `--Sequential message identifier, 0 to 3
#        | `--Sentence Number, 1 to 9
#         `--Total Number of sentences needed to transfer
   message, 1 to 9


# BBM - UAIS broadcast binary message
# Used to input NMEA format messages into AIS transponder
# for output as RF.
# For AIS Message Types 8, 14, 25, 26
```

```
#
# !AIBBM,x,x,x,x,x.x,s--s,x,*hh<CR><LF>
#        | | | |  |   |  |
#        | | | |  |   |  |
#        | | | |  |   |  |
#        | | | |  |   |  `--Number of fill bits (0-5)
#        | | | |  |    `--Encapsulated data
#        | | | |   `--VHF datalink message number (8, 14,
   see ITU-R M1371
#        | | |  `--AIS channel for broadcast of the radio
   message
#        | | `--Sequential message identifier, 0 to 9
#        | `--Sentence Number, 1 to 9
#         `--Total Number of sentences needed to transfer
   message, 1 to 9


import binascii
from argparse import ArgumentParser
import serial
import os
import time

############################################################
############################################################


def InputConversion(FN, srcMMSI, dstMMSI, chan, outfile):
    #This function converts and aligns the data strings
    #present in the input file to NMEA-0183 alphabet format.
    #Conversion to binary with 6-bit alignment and then to
    #NMEA-0183
```

```
# Parse Input
inStrings = []
print("Initiating file read, conversion, and
   transmission process...\r")

with open(FN, 'rb') as f:
    for line in f:

        # The following lines are a work around for to
        # retain the leading 0 in hex to binary
           conversion.
        # Python3 does not keep the leading zero. This
           is
        # necessary to ensure that information integrity
            is
        # retained through multiple conversion
           iterations.

        zero = b'0'
        line = zero + line.decode("utf-8-sig").encode("
           utf-8").strip()  # Removes UTF-8 BOM and \n
        line_binary = bin(int(binascii.hexlify(line),
           16))[8:]  # Converts each line to binary
        inStrings.append(line_binary)

for dataStrings in inStrings:

    #Calculate fill bits to maintain 6-bit "byte"
       alignment
    fill = 0

    binary_strings = dataStrings
```

```
        if len(dataStrings) %6 != 0:
            fill = 6 - (len(dataStrings) %6)
            dataStrings += "0"*fill

        #Convert to 6bit NMEA 0183 Alphabet format:
        NMEA_payload = Convert2NMEA0183(dataStrings)

        #Creation of !AIABM message
        createAIABM(NMEA_payload, fill, srcMMSI, dstMMSI,
           chan, binary_strings, outfile)

    return

#############################################################
#############################################################


def createAIABM(AIABMpayload, fill, src, dst, chan,
   binary_strings, outfile):
    print("Creating AIABM payload and sentence...\r")
    total = 1

    if len(AIABMpayload) > 82:
        total = int(len(AIABMpayload) / 82)

        if len(AIABMpayload) % 82 != 0:
            total += 1

    for i in range(total):
        sentence = "!AIABM," + str(total) + "," + str(i + 1)
            + "," + "0" + "," + dst + "," + chan + "," +
           "12" + "," + AIABMpayload + ","
```

```
        #This portion of code is designed to ensure that
        #the fill bit information is added on correctly
        #if there are more than two sentences. The fill
        #bit info needs to be on the last fragment of a
        #sentence series.

        if i == total - 1:
            sentence += str(fill)

        else:
            sentence += "0"

        sentence += "*" + checksum(sentence)
        print("AIABM Sentence creation successful. Ready for
            transmission...\r")
        transmit2Transponder(sentence, src, dst,
           binary_strings, outfile)

    return

############################################################
############################################################


def transmit2Transponder(txmsg, src, dst, binary_strings,
   outfile):
    serialStream = serial.Serial('/dev/ttyUSB0', baudrate
       =38400, timeout=1)
    print("Transmitting AIABM now...\r")

    if serialStream:
        serialStream.flushInput()
        serialStream.flushOutput()
```

```
        if True:
            txmsg = txmsg + "\r\n"
            print("Sending " + txmsg + " over serial now.\r\
               n")

            outputAIVDM(src, dst, binary_strings, outfile)

            serialStream.write(txmsg.encode())
            print("AIABM sent to transponder...\r\n")
            serialStream.close()



            time.sleep(1)



    else:
        print("Port is not open.\r\n")

    return

############################################################
############################################################


def checksum(sentence): # Adapted from Dosch-man's Blog.
                        # http://doschman.blogspot.com
                           /2013/01/calculating-nmea-
                           sentence-checksums.html
                        # Accessed: August 9, 2019
```

```
    print("Creating CHECKUM...\r")
    checksumdata = sentence[1:]  # drops the leading "!" or
       "$"
    csum = 0  # initializing first XOR value

    # For each char in checksumdata, XOR against the
       previous XOR's character.
    # The final XOR of the last char will be the checksum to
       verify against.

    for chk in checksumdata:
        csum ^= ord(chk)

    csum = str(hex(csum))[2:]
    print("CHECKSUM creation successful...\r\n")
    return csum

##############################################################
##############################################################


def Convert2NMEA0183(datastrings):
    print("Converting string data to NMEA-0183 6-bit format
       ...\r")
    NMEA0183_alphabet = "0123456789:;<=>?
      @ABCDEFGHIJKLMNOPQRSTUVW`abcdefghijklmnopqrstuvw"
    encode = ""

    for i in range(int(len(datastrings)/6)):
        k = i*6
        ch = int("00" + datastrings[k:k+6], 2)
        encode += NMEA0183_alphabet[ch]
```

```
    print("Conversion to NMEA-0183 6-bit format complete...\
        r\n")

    return encode

#############################################################
#############################################################


def outputAIVDM(srcMMSI, dstMMSI, payload, name):
    print("Initiating AIABM to AIVDM sentence conversion
        process for outfile...\r")
    zero = b'0'
    srcMMSI = srcMMSI.encode()
    srcMMSI = zero + srcMMSI
    dstMMSI = dstMMSI.encode()
    dstMMSI = zero + dstMMSI
    source_binary = str(bin(int(binascii.hexlify(srcMMSI),
        16))[8:])
    destination_binary = str(bin(int(binascii.hexlify(
        dstMMSI), 16))[8:])

    binaryInformation = '001100' + '00' + source_binary +
        '00' + destination_binary + '0' + '0' + payload
    AIVDM_fill = 0

    if len(binaryInformation) %6 != 0:
        AIVDM_fill = 6 - (len(binaryInformation) %6)
        binaryInformation += "0"*AIVDM_fill

    AIVDMmessagePayload = Convert2NMEA0183(binaryInformation
        )
```

```
    totalAIVDMlength = 1

    if len(AIVDMmessagePayload) > 82:
        totalAIVDMlength = int(len(AIVDMmessagePayload) /
            82)

        if len(AIVDMmessagePayload) % 82 !=0:
            totalAIVDMlength += 1


    createAIVDM_Msg(AIVDMmessagePayload, totalAIVDMlength,
        AIVDM_fill, name)

    return

#############################################################
#############################################################


def createAIVDM_Msg(AIVDMmessagePayload, totalAIVDMlength,
    AIVDM_fill, name):
    print("AIVDM message being created...\r")
    #Creates the actual AIVDM message to be stored in a file
        as reference for later signature verification

    for i in range(totalAIVDMlength):
        MsgPayload = AIVDMmessagePayload[(i * 82):(i * 82) +
            82]
        sentence = "!AIVDM," + str(totalAIVDMlength) + "," +
            str(i+1) + ",,A" + MsgPayload + ","

        if i == totalAIVDMlength - 1:
            sentence += str(AIVDM_fill)
```

```
        else:
            sentence += "0"

        sentence += "*" + checksum(sentence)

        with open('{}'.format(name, 'a+')) as f:
            f.writelines(str(time.time()) + "|" + sentence)
    print("AIVDM message created and written to file with
       timestamp...\r\n")
    return

#############################################################
#############################################################


if __name__ == "__main__":
    print("Script Initiating...\r")
    #Required Arguments

    parser = ArgumentParser(description="Converts user
       inputs to NMEA input messages to transponder for
       conversion and transmission.")
    parser.add_argument("AIS", nargs='*', help="input
       message")
    parser.add_argument("-f", "--file", dest="filename",
       nargs=1, type=str, help="read input from a file")
    parser.add_argument("-sm", "--source_mmsi", dest="
       source_mmsi", default="338999999", type=str, nargs=1,
        help="Input source mmsi. (Default: 338999999)")
    parser.add_argument("-dm", "--destination_mmsi", dest="
       destination_mmsi", default="366123000", type=str,
       nargs=1, help="Input destination mmsi (Default:
```

```
    366123000)")
parser.add_argument("-o", "--outfile", dest="outfile",
    nargs=1, type=str, help="specify a file to write
    sentences to.")
parser.add_argument("-c", "--chan", dest="chan", nargs
    =1, type=str, default="1", help="specify the AIS
    transmission channel. (Default\"A\")")  #0=No
    preference 1=A 2=B 3=Both

args = parser.parse_args()

#Check for minimum arguments
print("Checking for minimum argument length...\r")
if len(args.AIS) < 1:
    print("Addition arguments required.")
    quit()
print("Minimum argument length met...\r\n")

#Check if outfile currently exists
print("Checking if " + args.outfile[0] + " exists...\r")
if os.path.exists(args.outfile[0]):
    print(args.outfile[0] + " currently exists.
       Initiating deletion.\r")
    os.remove(args.outfile[0])  # Deletes outfile
    print(args.outfile[0] + " deleted.\r\n")

print(args.outfile[0] + " does not exist. It was either
    deleted or did not already exists...\r\n")


InputConversion(args.filename[0], args.source_mmsi[0],
    args.destination_mmsi[0], args.chan[0], args.outfile
    [0])
```

```
print("Script Finished...\r\n")
```

APPENDIX F:
InputEncoder.PY Process Flow Graph

This appendix features the processes and functions found within *InputEncoder.py*. This figure intends to logically organize the flow and construction of the Python source code.

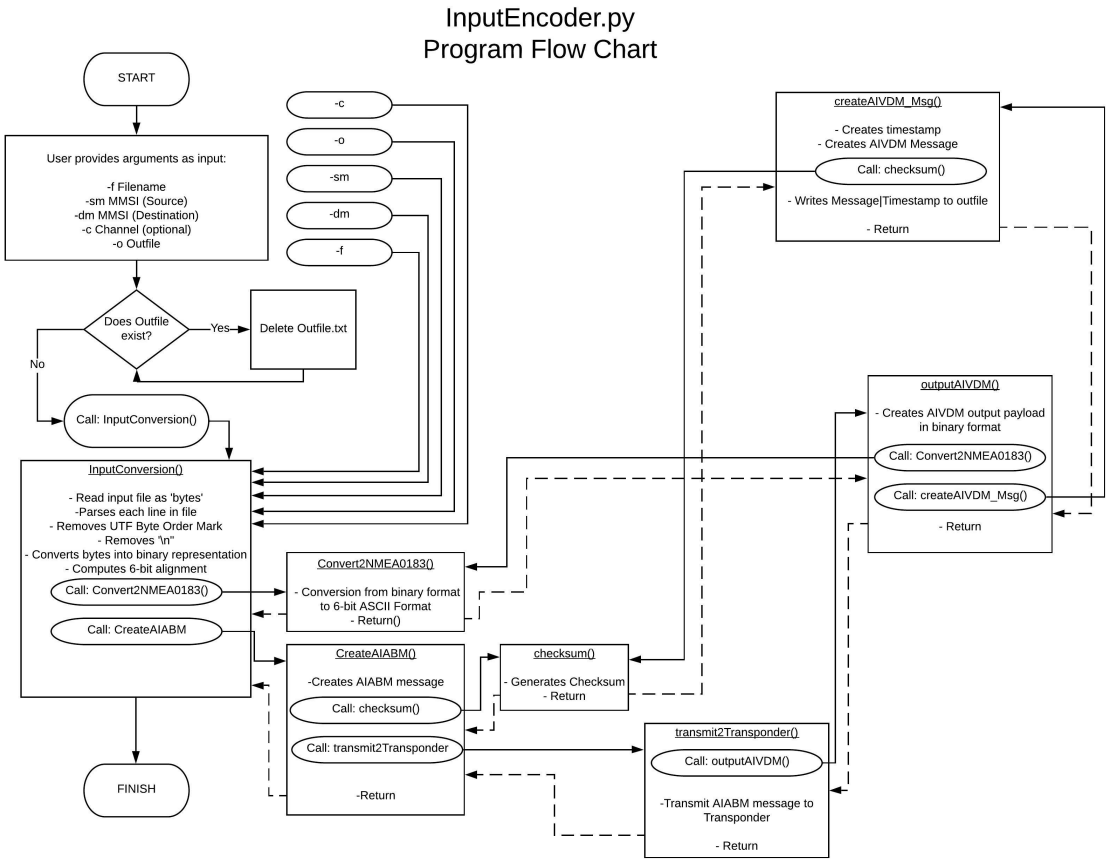


Figure F.1. *InputEncoder.py* Process Flow Graph

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G:
A_Challenger.PY

This appendix displays the code utilized for the CAMINO-701 proposed AIS system.

```
import serial
import queue
import multiprocessing
#import sys
from argparse import ArgumentParser
import binascii
from datetime import datetime

'''
from charm.toolbox.pairinggroup import PairingGroup,ZR,G1,G2
    ,pair
from charm.toolbox.iterate import dotprod
from charm.toolbox.hash_module import Waters
from charm.core.engine.util import *
'''

from charm.toolbox.pairinggroup import PairingGroup,ZR,G1,G2
    ,pair
from charm.core.engine.util import *
from charm.toolbox.IBSig import *


def convertToASCII(binary):
    print("Converting Digital Signature from binary to 8-bit
        ASCII...\r")
    return ''.join(chr(int(binary[i*8:i*8+8], 2)) for i in
       range(int(len(binary) / 8)))
```

```
def createAIVDMBinary(AIVDM):
    AIVDM = str(AIVDM.encode())[2:-5].encode()
    zero = b'0'
    line = zero + AIVDM
    print(line)
    line_binary = bin(int(binascii.hexlify(line), 16))[8:]

    return line_binary


def convertToBinary(ais_data):

    NMEA0183_alphabet = "0123456789:;<=>?
       @ABCDEFGHIJKLMNOPQRSTUVW`abcdefghijklmnopqrstuvw"

    decode = ""

    for c in ais_data:
        decode +='{0:b}'.format(NMEA0183_alphabet.find(c)).
           rjust(6, '0')
    return decode


#def SigVerify(MMSI, msg):
def SigVerify(msg, sig, MMSI):

    global time_dict

    timestamp = time_dict.get(msg)
    msg0 = msg.strip()
    msg = msg0 + str(timestamp)
```

```
'''
print("*****************************************\r")
print("DEBUG: TIMESTAMP FOR VERIFICATION: {0}\r".format(
    timestamp))
msg = msg + str(timestamp)
print(msg)
print("*****************************************\r")
'''

groupObj = PairingGroup('MNT201')
bls = BLS01(groupObj)

with open("publickey1.txt", "rb") as rfile1:
    g = rfile1.read()
    rfile1.close()

with open("publickey2.txt", "rb") as rfile2:
    gx = rfile2.read()
    rfile2.close()

print("Converting Digital Signature Elements from BYTE
   OBJECT to Pairing Element...\r")
sig = bytesToObject(sig, group)
g = bytesToObject(g, group)
gx = bytesToObject(gx, group)

print("Conversion sucessessful...\r")

assert bls.verify(g, gx, sig, msg), "Failure!!!\r"
print("Digital Signature Verification Successful...\r")
print("{0} is a valid message from {1} received
   originally at {2}\r\n".format(msg0, MMSI, timestamp))
```

```
    datetimeEND = datetime.now()
    ENDtime = datetimeEND.strftime("%d-%b-%Y (%H:%M:%S)")
    print("END TIME: {0}\r".format(ENDtime))

    return

    '''
    with open("masterpublic.txt", "rb") as rfile:
        public_bytes = rfile.read()
        rfile.close()
        print("Converting Digital Signature from BYTE OBJECT
            to Pairing Element...\r")
        signature = bytesToObject(sig, group)
        print("Conversion sucessessful...\r")
        master_public_key = bytesToObject(public_bytes,
           group)

        assert wat.verify(master_public_key, MMSI, msg,
           signature), "Invalid signature...\r"
        print("Digital Signature Verification Successful...\
           r\n")

    return
    '''


def convertTo6(msg):
    NMEA0183_alphabet = "0123456789:;<=>?
       @ABCDEFGHIJKLMNOPQRSTUVW`abcdefghijklmnopqrstuvw"
    encode = ""
    for i in range(int(len(msg)/6)):
        k = i*6
        ch = int("00" + msg[k:k+6], 2)
```

```
        encode += NMEA0183_alphabet[ch]

    return encode


def checksum(sentence):

    sentence = sentence[1:]
    sentence = sentence.encode("utf-8")

    csum = 0b0

    for i in range (len(sentence)):
        csum ^= sentence[i]

    csum = str(hex(csum).upper())[2:]

    if len(csum) != 2:
        csum = "0" + csum

    return csum



#ABM - Addressed binary and safety related message
#Used to input NMEA format messages into AIS transponder
#for output as RF.
#For AIS Message Types 6, 12, 25, 26
#
#!AIABM,x,x,x,xxxxxxxxx,x,x.x,s--s,x*hh<CR><LF>
#       | | |     |     |  |  |   |
#       | | |     |     |  |  |    `-Number of fill bits (0
   -5)
```

```
#       | | |     |     | |    '--Encapsulated data
#       | | |     |     | '--VHF datalink message number
   (6, 12, 25, 26), see ITU-R M1371
#       | | |     |      '--AIS channel for broadcast of the
    radio message (0-3)
#       | | |     '--The MMSI of destination AIS unit for
   the ITU-R M1371 message (9 digits; 30 bits)
#       | | '--Sequential message identifier, 0 to 3
#       | '--Sentence Number, 1 to 9
#        '--Total Number of sentences needed to transfer
   message, 1 to 9


#BBM - UAIS broadcast binary message
#Used to input NMEA format messages into AIS transponder for
    output as RF.
#For AIS Message Types 8, 14, 25, 26
#
#!AIBBM,x,x,x,x,xx,s--s,x*hh<CR><LF>
#       | | | |  |   |  |
#       | | | |  |   |  |
#       | | | |  |   |  |
#       | | | |  |   | '--Number of fill bits (0-5)
#       | | | |  |    '--Encapsulated data
#       | | | |  '--VHF datalink message number (8, 14, see
    ITU-R M1371
#       | | |  '--AIS channel for broadcast of the radio
   message
#       | | '--Sequential message identifier, 0 to 9
#       | '--Sentence Number, 1 to 9
#        '--Total Number of sentences needed to transfer
   message, 1 to 9
```

```
def createChallenge(payload, mmsi):
    # Create the number of NMEA senteces required to hold
      the AIS message
    print("##########################################\r")
    print("Initiating AIVDM to binary conversion process.\r
      ")
    payload_bin = createAIVDMBinary(payload)
    print("AIVDM to binary conversion process successful.\r
      ")
    print("Checking generated payload binary for 6-bit
      alignment.\r")

    total = 1
    fill = 0

    if len(payload_bin) % 6 != 0:
        fill = 6 - (len(payload_bin) % 6)
        payload_bin += "0" * fill

    print("Alignment complete.\r")
    print("Initiating conversion of challenge message
      payload to 6-bit ASCII.\r")
    AIABMpayload = convertTo6(payload_bin)
    print("Conversion to 6-bit ASCII complete.\r")
    print("Checking max 48 character payload size.\r")

    if len(AIABMpayload) > 48:
        total = int(len(AIABMpayload) / 48)

        if len(AIABMpayload) % 48 != 0:
            total += 1
    print("There are {0} total fragments.\r".format(total))
```

```
for i in range(total):

    if i != (total-1):

        AIABMpayload_send = AIABMpayload[(i*48):((i*48)
           + 48)]

        # Result will be in the form specified in the
        # NMEA 0183 ABM sentence
        sentence = "!AIABM," + str(total) + "," + str(i
           + 1) + "," + "0" + "," + str(mmsi) + "," +
           "1" + "," + "12" + "," + AIABMpayload_send +
           "," + "0"

    elif i == (total-1):

        AIABMpayload_send = AIABMpayload[(i * 48):]

        # Result will be in the form specified in the
        # NMEA 0183 ABM sentence
        sentence = "!AIABM," + str(total) + "," + str(i
           + 1) + "," + "0" + "," + str(mmsi) + "," +
           "1" + "," + "12" + "," + AIABMpayload_send +
           "," + str(fill)

    # calculate checksum for packet
    sentence += "*" + checksum(sentence)

    print("----------------------------------------\r")
    print("Sending challenge message to: {0}\r".format(
       mmsi))
    print("Challenge Message: {0}\r".format(sentence))
```

```
        print("Challenge Message Payload: {0}\r".format(
            AIABMpayload))
        print("Interrogated Message: {0}\r".format(payload))
        print("Transmitting challenge message to transponder
            .\r")
        transmit2Transponder(sentence)

    return


def transmit2Transponder(txmsg):
    print("Transponder has message for transmission.\r")
    serialStream = serial.Serial('/dev/ttyUSB1', baudrate
        =38400, timeout=1)

    if serialStream:

        serialStream.flushInput()
        serialStream.flushOutput()

        txmsg = txmsg + "\r\n"
        serialStream.write(txmsg.encode())
        serialStream.close()

    print("Message Transmission Complete...\n")
    print("-----------------------------------------\r\n")

    return



################################
####  CRYPTO PORTION START  ####
```

```
################################

class BLS01(IBSig):

    def __init__(self, groupObj):
        IBSig.__init__(self)
        global group
        group = groupObj


    def dump(self, obj):
        return objectToBytes(obj, group)


    def verify(self, g, gx, sig, message):
        M = self.dump(message)
        h = group.hash(M, G1)
        if pair(sig, g) == pair(h, gx):
            return True
        return False

    '''

#| From: "B. Waters - Efficient identity-based encryption
#| without random oracles"
#| Published in: EUROCRYPT 2005
#| Available from: Vol 3494 of LNCS, pages 320-329
#| Notes:

#* type:           signature (ID-based)
#* setting:        bilinear groups (asymmetric)

#:Authors:    J. Ayo Akinyele
```

```
#:Date:         11/2011

class WatersSig:

    def __init__(self, groupObj):
        global group,lam_func
        global waters
        global z
        global l

        z = 5
        l = 32
        group = groupObj
        lam_func = lambda i,a,b: a[i] ** b[i]
        waters = Waters(group, z, l)


    def sign(self, mpk, sk, M):
        m = waters.hash(M) # return list from m1,...,mz
        (k1, k2) = sk
        s  = group.random(ZR)
        S1 = k1 * ((mpk['u2t'] * dotprod(1, -1, mpk['z'],
           lam_func, mpk['u'], m)) ** s)
        S2 = k2
        S3 = mpk['g1'] ** -s

        return {'S1':S1, 'S2':S2, 'S3':S3}


    def verify(self, mpk, ID, M, sig):
        k = waters.hash(ID)
        m = waters.hash(M)
        (S1, S2, S3) = sig['S1'], sig['S2'], sig['S3']
```

```
        A, g2 = mpk['A'], mpk['g2']
        comp1 = dotprod(1, -1, mpk['z'], lam_func, mpk['ub
            '], k)
        comp2 = dotprod(1, -1, mpk['z'], lam_func, mpk['ub
            '], m)
        lhs = (pair(S1, g2) * pair(S2, mpk['u1b'] * comp1) *
             pair(S3, mpk['u2b'] * comp2))
        #if ((pair(S1, g2) * pair(S2, mpk['u1b'] * comp1) *
            pair(S3, mpk['u2b'] * comp2)) == A):

        if lhs == A:
            return True

        return False
    '''


    #assert wat.verify(master_public_key, MMSI, msg, sig), "
        invalid signature"


################################
####  CRYPTO PORTION END   ####
################################


def main(own_mmsi):
    global time_dict

    while not message_queue.empty():
        raw_message = message_queue.get()
        ais_data = raw_message.split(",")[5]  # AIVDM
            Payload
```

```
ais_admin = raw_message.split(",")[6]  # AIVDM Fill
    bit*Checksum
ais_fill = int(ais_admin.split("*")[0])  # AIVDM
    Fill bits
ais_data_bin = convertToBinary(ais_data)  # Converts
     AIVDM Payload to 6-bit binary
mmsi_bin = ais_data_bin[8:38]  # extracts mmsi
    binary
msgid_bin = ais_data_bin[0:6]  # extracts message id
     binary
mmsi = int(mmsi_bin, 2)  # converts binary to base
    10 int.
msgid = int(msgid_bin, 2)  # converts binary to base
     10 int.

print("Message from {0}\r".format(mmsi))
print("Message type: {0}\r".format(msgid))

print("************************************\r\n")
if msgid == 12 and mmsi != own_mmsi:
     print("####################################\r")
     print("Initiating AIVDM message analysis...\r")
     print("####################################\r\n")

     global challenged_message
     challenged_message = raw_message



     datetimeObj = datetime.now()
     received_time = datetimeObj.strftime("%d-%b-%Y
         (%H:%M) ")
     time_dict.update({raw_message: received_time})
```

```
            print("#####################################\r")
            print("Message received at/on: {0}\r".format(
               received_time))
            print("Message is from: {0}\r".format(mmsi))
            print("AIVDM Message Payload: {0}\r".format(
               ais_data))
            print("Automatically initiating AIVDM message
               challenge processes.\r")
            print("#####################################\r\n")

            print("Calling createChallenge().\r")
            combo = ais_data + "," + ais_admin
            createChallenge(combo, mmsi)
            print("Challenge message creation & transmission
                complete.\r\n")
            print("#####################################\r\n")

        elif msgid == 14 and mmsi != own_mmsi:
            print("\n")
            print("Response Message Detected.\r")
            signature_payload_bin = ais_data_bin[40:]

            if ais_fill != 0:
                ais_fill = -ais_fill
                signature_payload_bin =
                   signature_payload_bin[:ais_fill]

            signature_bin_queue.put(signature_payload_bin)


            if signature_bin_queue.full():
                totalbin = ""
```

```
                for i in range(5):
                    add = signature_bin_queue.get()
                    totalbin += add

                signature_payload_bin = totalbin

                signature_ASCII = convertToASCII(
                   signature_payload_bin)
                print("\r")
                print("********************************\r")
                print("Digital Signature:\r")
                print(signature_ASCII)
                print("********************************\r")


                print("Digital Signature to ASCII conversion
                    complete.\r")
                print("Initiating signature verification
                   process.\r")

                signature_ASCII = signature_ASCII.encode()
                print(signature_ASCII)
                #SigVerify(mmsi, challenged_message)
                SigVerify(challenged_message,
                   signature_ASCII, mmsi)



if __name__ == "__main__":
    time_dict = dict()
    challenged_message = ''
```

```
parser = ArgumentParser(description="Input one's own
   MMSI.")
parser.add_argument("-mmsi", "--mmsi", dest="mmsi",
   nargs=1, type=str, help="MMSI value")

args = parser.parse_args()

# Check for minimum arguments
if len(sys.argv) == 1:
    print("Input MMSI value\r")
    quit()

signature_bin_queue = queue.Queue(5)
message_queue = queue.Queue()
own_mmsi = args.mmsi[0]

while True:
    serialStream = serial.Serial('/dev/ttyUSB1',
       baudrate=38400, timeout=1)
    received = serialStream.readline().decode()

    if "!AIVDM" in received:
        print("\n")
        print("**********************************\r")
        print("AIVDM message received: {0}\r".format(
           received))


        message_queue.put(received)

        main_function = multiprocessing.Process(name='
           main_f', args=own_mmsi, target=main(own_mmsi)
           )
```

```
main_function.daemon = True
main_function.start()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H:
A_Challenger.PY Process Flow Graph

This appendix features the processes and functions found within *A_Challenger.py*. This figure intends to logically organize the flow and construction of the Python source code.

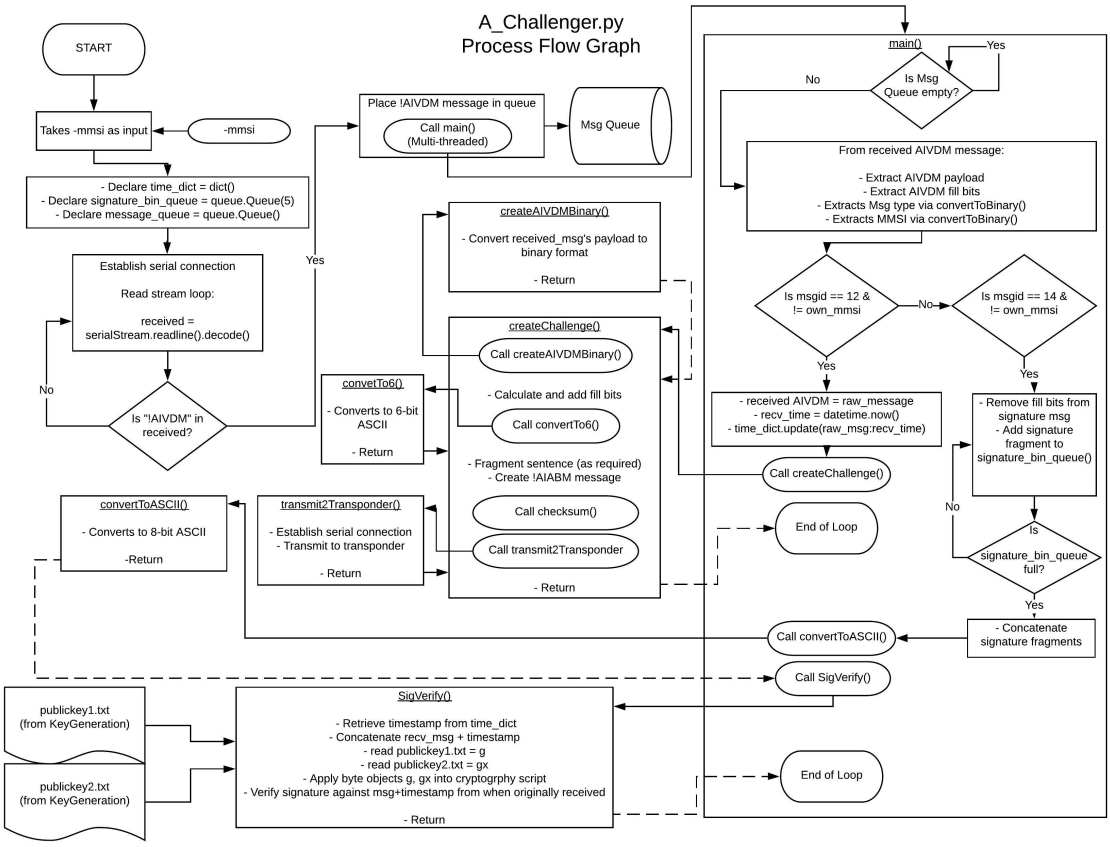


Figure H.1. *A_Challenger.py* Process Flow Graph

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I:
B_Challenger.PY

This appendix displays the code utilized for the FA-150 proposed AIS system.

```
import serial
import queue
import multiprocessing
import binascii
import sys
import random
import time
from argparse import ArgumentParser

'''
from charm.toolbox.pairinggroup import PairingGroup,ZR,G1,G2
    ,pair
from charm.toolbox.iterate import dotprod
from charm.toolbox.hash_module import Waters
'''
#This import is for BLS_04 Implementation
from charm.toolbox.pairinggroup import PairingGroup,ZR,G1,G2
    ,pair
from charm.toolbox.IBSig import *


from charm.core.engine.util import *

#ABM - Addressed binary and safety related message
#Used to input NMEA format messages into AIS transponder for
    output as RF.
#For AIS Message Types 6, 12, 25, 26
```

```
#
#!AIABM,x,x,x,xxxxxxxxx,x,x.x,s--s,x,*hh<CR><LF>
#       | | |     |     |  |  |   |
#       | | |     |     |  |  |   '-Number of fill bits (0
   -5)
#       | | |     |     |  |   '--Encapsulated data
#       | | |     |     |  '--VHF datalink message number
   (6, 12, 25, 26), see ITU-R M1371
#       | | |     |      '--AIS channel for broadcast of the
    radio message (0-3)
#       | | |     '--The MMSI of destination AIS unit for
   the ITU-R M1371 message (9 digits; 30 bits)
#       | | '--Sequential message identifier, 0 to 3
#       | '--Sentence Number, 1 to 9
#        '--Total Number of sentences needed to transfer
   message, 1 to 9


#BBM - UAIS broadcast binary message
#Used to input NMEA format messages into AIS transponder for
    output as RF.
#For AIS Message Types 8, 14, 25, 26
#
#!AIBBM,x,x,x,x,xx,s--s,x,*hh<CR><LF>
#       | | | |  |   |  |
#       | | | |  |   |  |
#       | | | |  |   |  |
#       | | | |  |   |  '--Number of fill bits (0-5)
#       | | | |  |    '--Encapsulated data
#       | | | |   '--VHF datalink message number (8, 14, see
    ITU-R M1371
#       | | |  '--AIS channel for broadcast of the radio
   message
```

```
#       | | '--Sequential message identifier, 0 to 9
#       | '--Sentence Number, 1 to 9
#        '--Total Number of sentences needed to transfer
   message, 1 to 9

def convertToASCII(binary):
    print("Converting NMEA-0183 6-bit binary to 8-bit ASCII
       .\r")
    return ''.join(chr(int(binary[i*8:i*8+8], 2)) for i in
       range(int(len(binary) / 8)))

def convertNMEAToNMEABinary(ais_data, ais_fill):
    # This function converts the 6-bit ASCII characters
    # into its 6-bit binary equivalent and take into
    # account fill bits
    NMEA0183_alphabet = "0123456789:;<=>?
       @ABCDEFGHIJKLMNOPQRSTUVW`abcdefghijklmnopqrstuvw"

    print(ais_data)

    decode = ""
    ais_fill = -abs(int(ais_fill))

    for c in ais_data:
        decode +='{0:b}'.format(NMEA0183_alphabet.find(c)).
           rjust(6,'0')

    if ais_fill != 0:
        decode = decode[:ais_fill]

    decode = decode[72:]
    return convertToASCII(decode)
```

```
def convertToBinary(ais_data):
    # This function converts the 6-bit ASCII characters
    # into its 6-bit binary equivalent
    NMEA0183_alphabet = "0123456789:;<=>?
       @ABCDEFGHIJKLMNOPQRSTUVW`abcdefghijklmnopqrstuvw"

    decode = ""

    for c in ais_data:
        decode +='{0:b}'.format(NMEA0183_alphabet.find(c)).
           rjust(6,'0')

    return decode

def checkfile(payload, ais_fill):
    # Must first convert received payload string back into
    # original NMEA-0183 6-bit prime.
    print("##########################################\r")
    print("The Challenge Message payload is currently in the
        wrong format.\r")
    print("Calling convertNMEAToNMEABinary().\r")
    lookup = convertNMEAToNMEABinary(payload, ais_fill)
    lookup = "!AIVDM,1,1,,A," + lookup
    print("Checking to see if {0} exists in outfile.\r".
       format(lookup))

    senttime = ""
    status = 0

    with open('Outfile.txt', 'r') as searchfile:

        for line in searchfile:
```

```
            line = line.rstrip()

            if lookup in line:
                print("{0} exists in outfile...\r".format(
                   lookup))
                senttime = line.split("|")[0]
                status = 1
                searchfile.close()
                return status, senttime, lookup


        print("{0} does not exist in outfile...\r".format(
           lookup))
        searchfile.close()
        return status, senttime, lookup




def SigCreation(msg, timestamp):
    print("##########################################\r")
    print("Initiating Digital Signature creation now.\r")
    print("##########################################\r\n")

    groupObj = PairingGroup('MNT159')
    #wat = WatersSig(groupObj)#For Waters Implementation
    bls = BLS01(groupObj)      #For BLS_04 Implemenation
    msg = msg+timestamp


    with open("privatekey.txt", "rb") as r:
        x = r.read()
        r.close()
```

```
'''
with open("publickey1.txt", "rb") as pk1r:
    g = pk1r.read()
    pk1r.close()

with open("publickey2.txt", "rb") as pk2r:
    gx = pk2r.read()
    pk2r.close()
'''



'''
with open("ShipONE.txt", "rb") as r:  #Ensure to change
   the filename to the appropriate AIS transponder
    one_bytes = r.read()
    r.close()

with open("masterpublic.txt", "rb") as rfile:
    public_bytes = rfile.read()
    rfile.close()
'''


xx = bytesToObject(x, group)

#master_public_key = bytesToObject(public_bytes, group)
   #For Waters Implementation
#sig = wat.sign(master_public_key, secret_key, msg)
   #For Waters Implementation

sig = bls.sign(xx, msg)

print("Signature creation successful...\r")
```

```
    sig = objectToBytes(sig, group)

    print("################################################\r")
    print("Signature Bytes: \r")
    print(sig)
    print("################################################\r")
    print("Signature Length: {0}\r\n".format(len(sig)))

    return sig

def convertTo6(msg):
    NMEA0183_alphabet = "0123456789:;<=>?
       @ABCDEFGHIJKLMNOPQRSTUVW`abcdefghijklmnopqrstuvw"
    encode = ""
    print("Response Message Payload Conversion initiating.\r
       ")

    for i in range(int(len(msg)/6)):
        k = i*6
        ch = int("00" + msg[k:k + 6], 2)
        encode += NMEA0183_alphabet[ch]

    print("Response Message Payload Conversion Successful.\r
       \n")
    return encode


def checksum(sentence):
    sentence = sentence[1:]
    sentence = sentence.encode("utf-8")
    csum = 0b0

    for i in range(len(sentence)):
```

```
        csum ^= sentence[i]

    csum = str(hex(csum).upper())[2:]

    if len(csum) != 2:
        csum = "0" + csum

    return csum


def createResponse(NMEApayload, fill):
    # Create the number of NMEA sentences required to hold
      the AIS message
    print("Response Message Creation Initiating.\r")
    print("Checking maximum size to not exceed 57 characters
      .\r")

    total = 1

    if len(NMEApayload) > 57:
        total = int(len(NMEApayload) / 57)


        if len(NMEApayload) % 57 != 0:
            total += 1


            print("AIBBM message exceeds 57 characters.\r")


    seq_num =  str(random.randint(0, 10))

    for i in range(total):
```

```
if i != (total-1):
    NMEApayload_send = NMEApayload[(i * 57):((i *
       57) + 57)]

    # Result will be in the form specified in the
       NMEA 0183 ABM sentence
    sentence = "!AIBBM," + "1" + "," + "1" + "," +
       "0" + "," + "1" + "," + "14" + "," +
       NMEApayload_send + "," + "0"
    #sentence = "!AIBBM," + str(total) + "," + str(i
        + 1) + "," + seq_num + "," + "1" + "," +
       "14" + "," + NMEApayload_send + "," + "0"
elif i == (total-1):
    NMEApayload_send = NMEApayload[(i * 57):]
    sentence = "!AIBBM," + "1" + "," + "1" + "," +
       "0" + "," + "1" + "," + "14" + "," +
       NMEApayload_send + "," + str(fill)
    #sentence = "!AIBBM," + str(total) + "," + str(i
        + 1) + "," + seq_num + "," + "1" + "," +
       "14" + "," + NMEApayload_send + "," + str(
       fill)




# calculate checksum for packet
sentence += "*" + checksum(sentence)
print("Broadcasting challenge message: " + "\r")
print(sentence)
print("Response Message Creation Successful...\r")
print("Sending Response Message to Transponder...\r\
   n")
```

```
        transmit2Transponder(sentence)
        time.sleep(4)

    return

def transmit2Transponder(txmsg):
    serialStream = serial.Serial('/dev/ttyUSB0', baudrate
       =38400, timeout=1)

    if serialStream:

        serialStream.flushInput()
        serialStream.flushOutput()

        txmsg = txmsg + "\r\n"
        serialStream.write(txmsg.encode())
        serialStream.close()

    print("Message Transmission Complete...\n")

    return



##################################
####   CRYPTO PORTION START  ####
##################################
'''
#| From: "B. Waters - Efficient identity-based encryption
   without random oracles"
    #| Published in: EUROCRYPT 2005
    #| Available from: Vol 3494 of LNCS, pages 320-329
    #| Notes:
```

```
    #* type:           signature (ID-based)
    #* setting:        bilinear groups (asymmetric)

    #:Authors:     J. Ayo Akinyele
    #:Date:        11/2011

class WatersSig:

    def __init__(self, groupObj):
        global group,lam_func
        global waters
        global z
        global l

        z = 5
        l = 32
        group = groupObj
        lam_func = lambda i,a,b: a[i] ** b[i]
        waters = Waters(group, z, l)


    def sign(self, mpk, sk, M):
        m = waters.hash(M) # return list from m1,...,mz
        (k1, k2) = sk
        s  = group.random(ZR)
        S1 = k1 * ((mpk['u2t'] * dotprod(1, -1, mpk['z'],
           lam_func, mpk['u'], m)) ** s)
        S2 = k2
        S3 = mpk['g1'] ** -s

        return {'S1':S1, 'S2':S2, 'S3':S3}
'''
```

```
class BLS01(IBSig):
    """
    >>> from charm.toolbox.pairinggroup import PairingGroup
    >>> group = PairingGroup('MNT224')
    >>> messages = { 'a':"hello world!!!" , 'b':"test
       message" }
    >>> ib = BLS01(group)
    >>> (public_key, secret_key) = ib.keygen()
    >>> signature = ib.sign(secret_key['x'], messages)
    >>> ib.verify(public_key, signature, messages)
    True
    """

    def __init__(self, groupObj):
        IBSig.__init__(self)
        global group
        group = groupObj


    def dump(self, obj):
        return objectToBytes(obj, group)


    def sign(self, x, message):
        M = self.dump(message)
        return group.hash(M, G1) ** x
##############################
####  CRYPTO PORTION END  ####
##############################
```

```
def main(own_mmsi):

    while not message_queue.empty():
        raw_message = message_queue.get()
        ais_data = raw_message.split(",")[5]     # AIVDM
           Payload
        ais_admin = raw_message.split(",")[6]    # AIVDM Fill
            bit*Checksum
        ais_fill = ais_admin.split("*")[0]       # AIVDM Fill
            bits
        ais_data_bin = convertToBinary(ais_data)  # Converts
            AIVDM Payload to 6-bit binary
        mmsi_bin = ais_data_bin[8:38]            # extracts
           mmsi binary
        msgid_bin = ais_data_bin[0:6]            # extracts
           message id binary
        mmsi = int(mmsi_bin, 2)                  # converts
           binary to base 10 int.
        msgid = int(msgid_bin, 2)                # converts
           binary to base 10 int.

        print("Message from {0}\r".format(mmsi))
        print("Message type: {0}\r".format(msgid))

        if msgid == 12 and mmsi != own_mmsi:
            print("**********************************\r\n")
            print("##################################\r")
            print("Challenge Message Detected\r")
            print("Challenge Message from: {0}\r".format(
               mmsi))
            print("Challenge Message Payload: {0} \r\n".
               format(ais_data)) # This is the AIVDM that
               was sent from encoder.py
```

```
print("####################################\r\n")

STATUS, SENTTIME, LOOKUP = checkfile(ais_data,
    ais_fill)  # This function checks the
    transmission file to locate
                                    # the
                                       raw_message.

if STATUS == 1:  # STATUS = 1; signifies that
    the message was sent out
                 # previously.
                 # Digital Signature Creation
                    Portion:
     zero = b'0'
     signature = SigCreation(LOOKUP, SENTTIME)  #
         Combination of the message in challenge
       & the sent time

     signature = zero + signature
     # convert to binary representation
     sig_bin = bin(int(binascii.hexlify(signature
       ), 16))[8:]

     # Calculate fill bits to maintain 6-bit byte
         alignment
     fill = 0

     if (len(sig_bin)) % 6 != 0:
          fill = 6 - (len(sig_bin) % 6)
          sig_bin += "0" * fill

     # Convert to 6bit NMEA 0183 alphabet
     responsePayload = convertTo6(sig_bin)
```

```
                createResponse(responsePayload, fill)  #
                    Creates the Broadcasted response message

        else:
            print("No additional actions taken.")
            print("**********************************\r\n")

if __name__ == "__main__":

    parser = ArgumentParser(description="Input one's own
        MMSI.")
    parser.add_argument("-mmsi", "--mmsi", dest="mmsi",
        nargs=1, type=str, help="MMSI value")

    args = parser.parse_args()

    # Check for minimum arguments
    if len(sys.argv) == 1:
        print("Input MMSI value\r")
        quit()

    message_queue = queue.Queue()
    own_mmsi = args.mmsi[0]
    while True:
        serialStream = serial.Serial('/dev/ttyUSB1',
            baudrate=38400, timeout=1)
        received = serialStream.readline().decode()

        if "!AIVDM" in received:

            print("**********************************\r")
            print("AIVDM message received.")
```

```
print(received)


message_queue.put(received)

main_function = multiprocessing.Process(name='
    main_f', args=own_mmsi, target=main(own_mmsi)
    )
main_function.daemon = True
main_function.start()
```

APPENDIX J:
B_Challenger.PY Process Flow Graph

This appendix features the processes and functions found within *B_Challenger.py*. This figure intends to logically organize the flow and construction of the Python source code.

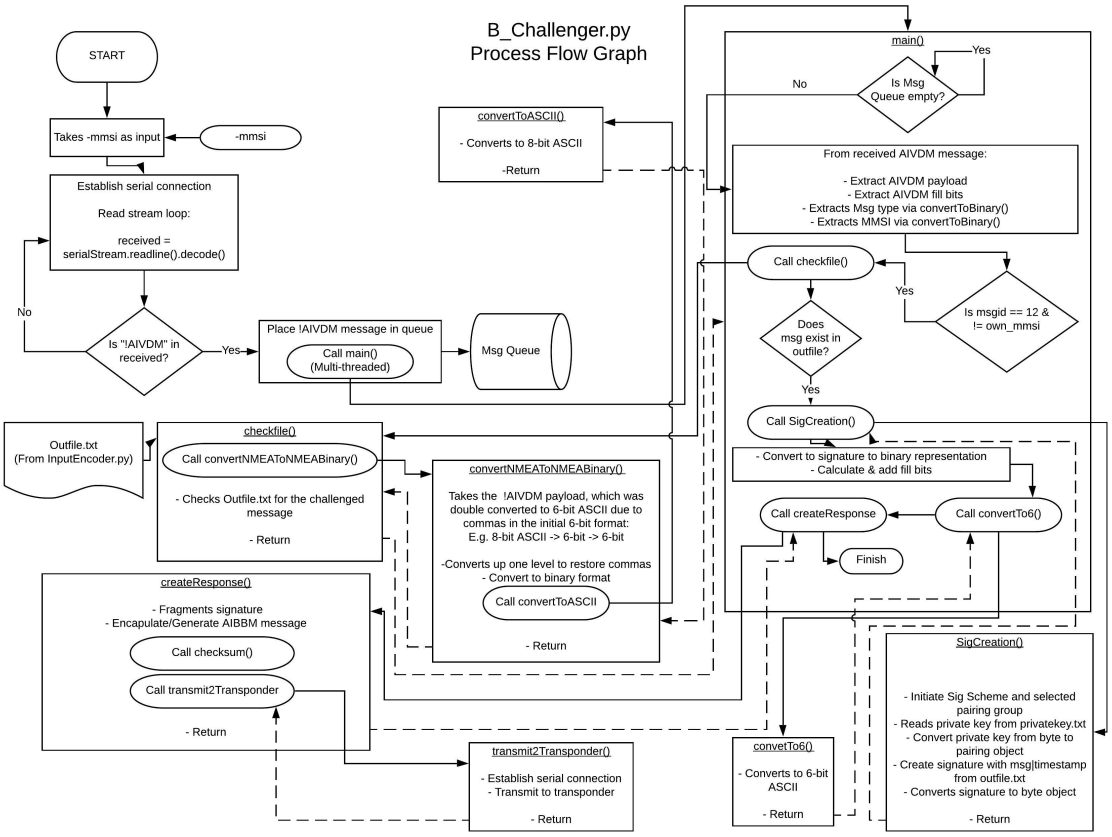


Figure J.1. *B_Challenger.py* Process Flow Graph

“

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX K:
A_Challenger.PY CPU Load Data

This table displays the measured system metrics of the proposed AIS authentication system. The data was captured over 30 mins at 10 second intervals on the CAMINO-701's laptop.

Table K.1. *A_Challenger.py*'s Measured CPU Performance Over Time

Passive	Idle	Active		Passive	Idle	Active	
Seconds	CPU Load Percentages			Average:	1.0	0.6	3.1
0	3	1.5	1.9	Mode:	0.2	0.4	0.4
10	0.4	1.2	1.2	Median:	0.6	0.4	2.7
20	1.1	0.4	0.6	Std:	1.1	0.9	3.0
30	0.2	0.1	3.4	Var:	1.3	0.8	9.0
40	2.9	0.6	8.7	Min:	0.0	0.0	0.1
50	1.6	0.6	1.7	Max:	6.1	7.5	17.2
60	0.4	0.3	3.4				
70	0.1	0.3	1.1				
80	0.7	0.3	0.2				
90	2.1	0.3	5.3				
100	0.4	0.7	1.8				
110	0.1	0.1	1.2				
120	0.3	0.4	0.6				
130	0.1	0.4	0.2				
140	2.9	0.4	0.2				
150	1.2	0.2	5.3				
160	0.4	0.4	9.4				
170	0.9	0.3	2.5				
180	0.3	0.5	1.2				
190	0.6	0.3	3.4				
200	0.8	0.7	5.3				
210	0.5	0.8	2.3				

Table K.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
220	4.4	0.4	4			
230	0.8	0.2	0.5			
240	1.2	0.4	0.9			
250	0.1	0.3	3.2			
260	0.7	0.6	5.1			
270	0.3	1	1.2			
280	0.7	1.3	3.4			
290	0.5	0.1	0.4			
300	0.8	0.5	0.7			
310	1.4	0.4	3.3			
320	0.8	0.3	11.4			
330	0.3	0.2	1			
340	1	0.4	4.1			
350	0.5	0.3	0.4			
360	1.2	0.3	1.2			
370	0.5	0.3	3			
380	0.9	0.4	12			
390	0.1	0.2	1.4			
400	3.1	0.3	4.1			
410	0.9	0.3	0.6			
420	0.6	0.5	5.5			
430	0.2	0.2	3.4			
440	0.7	0.5	10.9			
450	0.2	0.2	0.7			
460	0.5	0.3	4			
470	0.2	0.2	0.3			
480	0.6	0.6	1			
490	0.3	0.3	3.1			
500	3.5	0.6	5.5			
510	1.8	0.2	1			
520	0.7	0.6	3.3			

Table K.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
530	0.3	0.5	0.4			
540	0.6	0.6	0.9			
550	0.2	0.5	2.7			
560	0.7	0.3	12.2			
570	0.2	0.4	1.2			
580	4.5	1.1	4.1			
590	1.1	4.7	0.4			
600	1	0.4	0.8			
610	0.9	0.5	3.3			
620	6	0.9	5.6			
630			0.9			
640	1.3	0.9	3.7			
650	1	1.1	0.4			
660	0.3	0.2	0.5			
670	0.8	0.8	2.9			
680	0.4	0.3	5.3			
690	1.1	0.7	1.2			
700	4.1	0.3	3.4			
710	0.2	0.6	0.3			
720	0.1	0.6	1.8			
730	0.1	0.6	2.8			
740	0.3	0.5	17.2			
750	0.3	0.7	0.8			
760	2.6	0.3	4.3			
770	0.7	0.4	0.3			
780	0.1	0.1	4.3			
790	0.2	1	2.8			
800	0.2	0.8	5.3			
810	0.2	0.7	0.8			
820	0.2	0.2	3			
830	3	0.3	0.5			

Table K.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
840	0.3	0.2	0.6			
850	1.4	0.5	2.8			
860	3	0.4	11.7			
870	1.1	0.7	0.6			
880	0.6	0.6	4.3			
890	0.3	0.4	0.3			
900	0.6	0.2	0.7			
910	0.2	0.5	2.9			
920	0.6	0	12.9			
930	0.5	0.7	1.2			
940	5.8	1	4.3			
950	1.6	7.1	0.4			
960	3.6	0.5	0.9			
970	2.8	0	2.4			
980	6.1		5			
990	0.4	1.2	1.3			
1000	0.6	7.5	3.3			
1010	0.3	1	0.4			
1020	0.4	3	2			
1030	0.1	0.7	2.7			
1040	1.1	3.3	5.1			
1050	0.2	0.2	1.1			
1060	0.4	0.8	3.7			
1070	0.3	0	0.1			
1080	0.6	0.4	0.8			
1090	0.7	0.4	2.9			
1100	0.8	0.6	5.2			
1110	1	0.2	2.4			
1120	4	0.6	3.5			
1130	1.2	0.4	0.7			
1140	0.5	0.5	3.7			

Table K.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
1150	0.4	0.4	2.3			
1160	0.5	0.6	7			
1170	0.5	0.1	1			
1180	0.8	0.5	5.1			
1190	0.2	0.2	4			
1200	0.6	0.4	1.3			
1210	0.1	0.3	3.5			
1220	3.4	0.7	4.9			
1230	0.9	0.4	1			
1240	0.6	0.5	3.4			
1250	2.3	0.3	0.5			
1260	0.7	0.3	0.4			
1270	0.1	0.1	2.9			
1280	0.6	0.7	5			
1290	0.4	0.5	1.2			
1300	2.8	0.4	4.9			
1310	1	0.2	0.7			
1320	1.3	0.5	1.7			
1330	0.3	0.3	6.4			
1340	0.8	0.7	10.7			
1350	0.3	0.3	1.7			
1360	0.5	0.5	3.2			
1370	0.2	0.2	0.7			
1380	0.4	0.5	0.2			
1390	0.4	0.2	3.3			
1400	0.6	0.6	12			
1410	0.2	0.2	1.7			
1420	0.7	0.4	3.7			
1430	0.3	0.6	0.5			
1440	0.4	0.4	0.5			
1450	0.5	0.1	3			

Table K.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
1460	1.2	0.7	10.9			
1470	0.2	0.2	1.6			
1480	3.7	0.5	3.5			
1490	1.2	0.2	0.5			
1500	0.7	0.7	0.5			
1510	0.2	0	4.6			
1520	0.8	0.7	5.2			
1530	0.3	0.3	0.9			
1540	0.5	0.4	3.7			
1550	0.1	0.2	3.7			
1560	0.8	0.5	1.1			
1570	0.2	0.1	3.5			
1580	3.4	0.8	11.1			
1590	0.9	0.3	1.3			
1600	0.7	0.6	5			
1610	0.5	0.2	5.5			
1620	0.2	0.4	1			
1630	0.7	0.2	3.3			
1640	0	0.7	0.4			
1650	0.7	0.1	0.9			
1660	0.3	0.5	0.4			
1670	0.7	0.1	0.1			
1680	0.1	0.4	0.7			
1690	0.6	0	0.5			
1700	0.4	0.8	4.6			
1710	0.5	0.2	5.2			
1720	0.4	0.4	1.9			
1730	0.6	0.2	3.7			
1740	0.4	0.4	3.6			
1750	0.4	0.2	5.7			
1760	0.6	0.7	3.5			

Table K.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
1770	0.2	0.3	7.1			
1780	0.7	0.9	4.2			

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX L:
B_Challenger.PY CPU Load Data

This table displays the measured system metrics of the proposed AIS authentication system. The data was captured over 30 mins at 10 second intervals on the FA-150's laptop.

Table L.1. *B_Challenger.py*'s Measured CPU Performance Over Time

Passive	Idle	Active		Passive	Idle	Active	
Seconds	CPU Load Percentages			Average:	1.9	1.0	3.7
10	2.4	0.9	4.5	Mode:	1.4	1.2	2.4
20	1.3	1.3	5.8	Median:	1.6	1.1	3.3
30	2.6	2.3	3.9	Std:	1.1	0.6	2.3
40	1.1	0.7	6.9	Var:	1.3	0.4	5.1
50	1.5	1.4	5.9	Min:	0.5	0.2	0.8
60	1.9	1.2	3.7	Max:	9.7	6.1	24.1
70	4.5	1.3	2.4				
80	2.3	1.3	2.4				
90	6.7	2.1	2.4				
100	4.3	1.5	2.4				
110	2.2	1.8	5.2				
120	2.3	0.4	3.3				
130	2.2	1.4	2.2				
140	1.9	0.9	2				
150	1.3	1.7	2.4				
160	1.8	0.3	2.8				
170	2.4	1.1	5.2				
180	2.1	0.4	3.3				
190	2.3	1.1	4.2				
200	3.2	1.1	2.6				
210	2.2	1.4	3.3				
220	1.9	0.5	2.2				

Table L.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
230	1.7	1.2	4.2			
240	1.4	0.4	2.2			
250	2.1	1.2	1.7			
260	2.1	1.4	2.1			
270	2	1.8	2.5			
280	5.1	0.3	2.2			
290	1.3	1.3	4.2			
300	0.9	0.5	3.1			
310	1.7	1.2	2.3			
320	1.9	1	2.4			
330	3.2	1.7	2.6			
340	1	0.6	5.4			
350	1	1.4	5.4			
360	1	0.4	3.1			
370	1.6	1.3	2.5			
380	6.3	1.3	2.1			
390	2.2	1.4	2			
400	1.2	6.1	3.6			
410	1.2	1.2	3.6			
420	1.7	1.1	5.3			
430	2.2	0.4	2.3			
440	2	2.1	2.3			
450	1.8	0.5	2.3			
460	1.5	1	2.3			
470	2.4	0.5	4			
480	1.4	1.2	5.5			
490	1.5	1.2	2.4			
500	2.1	1.7	2.1			
510	2.4	0.3	2.4			
520	1.9	1.2	2.4			
530	1.4	0.4	3.5			

Table L.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
540	1.4	1.3	3.4			
550	1.3	1.7	2.3			
560	1.7	1.5	1.9			
570	4.7	0.5	1.9			
580	1.1	1.2	2.8			
590	1.5	0.4	3.8			
600	1.6	1.3	5.6			
610	1.2	1.6	9.8			
620	2.6	1.8	4.1			
630	1.2	0.2	4.5			
640	1.7	1.3	3			
650	1.7	0.2	2.7			
660	2.8	1.2	6.2			
670	0.7	1.1	5.6			
680	2.3	1.7	5.4			
690	2.6	0.3	3.9			
700	0.9	1	3.1			
710	1.1	0.5	3.1			
720	1.5	0.8	2.9			
730	1.5	1.2	5			
740	2.2	1.4	4.2			
750	2.2	0.5	2.9			
760	1.4	1.3	3.4			
770	2.1	0.3	13.1			
780	2	1	3			
790	1.3	1.1	3.9			
800	1.8	1.2	2.9			
810	1.7	0.6	2.9			
820	1.6	1.1	3.6			
830	1.9	0.4	2.8			
840	1.7	1.4	3.4			

Table L.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
850	1.2	1.5	4.2			
860	2.3	1.3	2.5			
870	5	0.4	3.3			
880	1.6	1	3			
890	1.5	0.3	3.5			
900	1.3	1.1	3.2			
910	1.2	0.8	3.6			
920	3.7	1.5	2.6			
930	0.8	0.4	0.9			
940	1.5	1.3	0.8			
950	2	0.4	2.9			
960	1.2	1.2	3.6			
970	1.4	1.2	5.3			
980	1.8	1.3	3.9			
990	4.3	0.4	3			
1000	1	1.2	2.5			
1010	0.6	0.4	3.4			
1020	1.1	1.5	3.9			
1030	1.8	1.4	5.4			
1040	0.8	0.9	3.8			
1050	0.8	0.4	3.6			
1060	1.6	1.2	2.9			
1070	1	0.3	8.3			
1080	0.5	0.9	3.8			
1090	1.4	1.5	3.5			
1100	0.9	2.2	3.3			
1110	1.7	0.6	3			
1120	2.1	1.3	2.3			
1130	1.3	0.3	3.8			
1140	3.3	1.3	3.5			
1150	4.5	1.3	4.5			

Table L.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
1160	1.1	0.8	4.4			
1170	1.2	0.4	24.1			
1180	1.1	1	3.3			
1190	1.2	0.2	3.8			
1200	1.7	1.1	3.1			
1210	1.9	1.8	3.4			
1220	0.9	1.2	3.2			
1230	1.4	0.4	3.2			
1240	1.2	1.2	2.1			
1250	0.9	0.3	2.8			
1260	1.4	1.2	3.3			
1270	1.3	1.2	3.6			
1280	0.8	1.3	3.4			
1290	1.3	0.3	3.1			
1300	1	1.3	2.9			
1310	1.1	0.2	3.6			
1320	3.5	1.2	3			
1330	4.1	1.2	3.6			
1340	2.2	0.5	4.5			
1350	2.8	0.3	4.8			
1360	2.1	1.1	2.6			
1370	0.9	0.5	9.4			
1380	2.6	1.2	3.2			
1390	2.2	1.6	3.8			
1400	1.5	0.9	3.4			
1410	1.8	0.6	3.2			
1420	1.8	1	0.8			
1430	1.4	0.4	1.5			
1440	1.5	1.2	3.7			
1450	9.7	1.7	5			
1460	1.1	1.1	4.6			

Table L.1 continued from previous page

	Passive	Idle	Active	Passive	Idle	Active
1470	1.6	0.4	3.7			
1480	1.8	1.1	3.4			
1490	1.1	0.3	3.7			
1500	2.2	1.3	2.8			
1510	1.4	1.2	5.7			
1520	1	1.3	3.7			
1530	1.1	0.3	2			
1540	1.5	1.4	2.6			
1550	1.4	0.3	2.9			
1560	2.4	1.2	2.6			
1570	1.1	1.7	4.2			
1580	1.3	1.1	2.8			
1590	3.5	0.4	3.2			
1600	3	1.1	2.2			
1610	1.6	0.3	3.3			
1620	2.5	1	3			
1630	1.5	1.3	4.1			
1640	1.1	1	3.5			
1650	2.4	0.4	2.5			
1660	1.4	1.1	2.1			
1670	1.6	0.3	12.9			
1680	2.1	1.3	3			
1690	1.9	1.5	4.4			
1700	1.5	1.3	2.7			
1710	1.3	0.4	2.3			
1720	1.3	2	2.1			
1730	1.4	0.5	3.9			
1740	1.8	1.5	3			
1750	3.5	1.3	3.8			
1760	2.4	1.2	3.4			
1770	1.4	0.3	2.9			

Table L.1 continued from previous page

	Passive	Idle	Active		Passive	Idle	Active
1780	1.4	1.1	4.4				

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX M:
IBC_BLS04.PY

This appendix displays our work and construction of an identity-based digital signature based on the arithmetic proofs discussed in [40].

```
from charm.toolbox.pairinggroup import PairingGroup,ZR,G1,G2
    ,pair
from charm.core.engine.util import objectToBytes
from charm.toolbox.IBSig import *

debug = False

class BLS01(IBSig):

    def __init__(self, groupObj):
        IBSig.__init__(self)
        global group
        group = groupObj


    def dump(self, obj):
        return objectToBytes(obj, group)


    def setup(self):
        g, x = group.random(G2), group.random()
        g_x = g ** x
        mpk = {'g^x':g_x, 'g':g}
        msk = {'x':x}
        return (mpk, msk)
```

```
    def keygen(self, MMSI, x):
        pk = group.hash(MMSI, G1)
        sk = pk ** x
        return (pk, sk)


    def sign(self, message, sk, g):
        M = self.dump(message)
        alpha = group.random()
        S = sk + (group.hash(M, G1) ** alpha)
        T = g ** alpha

        return {'S1':S, 'S2':T}


    def verify(self, message, S, g, pk, g_x, T):
        M = self.dump(message)
        h = group.hash(M, G1)
        if pair(S, g) == (pair(pk, g_x) * pair(h, T)):
            return True
        return False


def main():
    groupObj = PairingGroup('MNT159')
    MMSI = "338999999"
    m = "HELLO WORLD"
    bls = BLS01(groupObj)

    (mpk, msk) = bls.setup()

    (pk, sk) = bls.keygen(MMSI, msk['x'])
```

```
    sig = bls.sign(m, sk, mpk['g'])
    sig_bytes = objectToBytes(sig, group)
    print(len(sig_bytes))


    if debug: print("Message: '%s'" % m)
    if debug: print("Signature: '%s'" % sig)
    assert bls.verify(m, sig['S1'], mpk['g'], pk, mpk['g^x
        '], sig['S2']), "Failure!!!"
    if debug: print('SUCCESS!!!')


if __name__ == "__main__":
    debug = True
    main()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX N:
CertificatelessPKI.PY

This appendix details our work and construction of a certificateless digital signature from [37].

```
from charm.toolbox.pairinggroup import PairingGroup,ZR,G1,G2
    ,pair
from charm.core.engine.util import objectToBytes
from charm.toolbox.IBSig import *

debug = False

class CertificatelessPKI(IBSig):


    def __init__(self, groupObj):
        IBSig.__init__(self)
        global group
        group = groupObj


    def setup(self):
        P = group.random(G2)
        s = group.random()
        P_0 = P ** s
        return s, P_0, P


    def PartialPrivateKeyExtract(self, MMSI, s, P, P_0):
        Q_A = group.hash(MMSI, G1)
        D_A = Q_A ** s
```

```
    #validity checks
    if pair(D_A, P) == pair(Q_A, P_0):
        print("Partial Private Key Extract is correct\n
            ")
    return D_A, Q_A


def SetSecretValue(self):
    x_A = group.random()
    return x_A


def SetPrivateKey(self, D_A, x_A):
    S_A = D_A ** x_A
    return S_A


def SetPublicKey(self, x_A, P, P_0):
    X_A = P ** x_A
    Y_A = P_0 ** x_A
    return X_A, Y_A


def Sign(self, P, M, S_A):
    a = group.random()
    r = pair((P ** a), P)
    v = group.hash(M, r)
    U = (S_A ** v) + (P ** a)
    return U, v


def Verify(self, X_A, P_0, Y_A, P, v, U, M, Q_A):
```

```
        if pair(X_A, P_0) == pair(Y_A, P):
            print('First Verification Holds True\n')

        r = pair(U,P) * (pair(Q_A, -Y_A)) ** v
        if v == group.hash(M, r):
            return True
        return False

def main():
    groupObj = PairingGroup('SS512')
    MMSI = "338999999"
    M = "HELLO WORLD"
    bls = CertificatelessPKI(groupObj)

    (s, P_0, P) = bls.setup()
    print("debug 1\r")
    D_A, Q_A = bls.PartialPrivateKeyExtract(MMSI, s, P, P_0)
    print("debug 2\r")
    x_A = bls.SetSecretValue()
    print("debug 3\r")
    S_A = bls.SetPrivateKey(D_A, x_A)
    print("debug 4\r")
    X_A, Y_A = bls.SetPublicKey(x_A, P, P_0)
    print("debug 5\r")
    U, v = bls.Sign(P, M, S_A)
    print("debug 6\r")
    assert bls.Verify(X_A, P_0, Y_A, P, v, U, M, Q_A), "
       Failure!!!"



if __name__ == "__main__":
    main()
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] M. Harris. (2019, Nov.). Ghost ships, crop circles, and soft gold: A GPS mystery in Shanghai. MIT Technology Review. [Online]. Available: https://www.technologyreview.com/s/614689/ghost-ships-crop-circles-and-soft-gold-a-gps-mystery-in-shanghai/. Accessed Nov. 21, 2019.

[2] M. Balduzzi, A. Pasta, and K. Wilhoit, "A security evaluation of ais automated identification system," in *Proceedings of the 30th Annual Computer Security Applications Conference* (ACSAC '14). New York, NY, USA: ACM, 2014, pp. 436–445. Available: http://doi.acm.org/10.1145/2664243.2664257

[3] About the industry. (2019). World Shipping Council. [Online]. Available: http://www.worldshipping.org/about-the-industry. Accessed Apr. 29, 2019.

[4] Trade statistics. (2019). World Shipping Council. [Online]. Available: http://www.worldshipping.org/about-the-industry/global-trade/trade-statistics. Accessed Apr. 29, 2019.

[5] W. S. Council. (2016, Dec. 16). Top container ship trade routes. *Geopolitical Futures*. [Online]. Available: https://geopoliticalfutures.com/top-container-ship-trade-routes/. Accessed: 2019-04-29.

[6] SOLAS Chapter 5, Regulation 19.2. (2019). International Maritime Organization. [Online]. Available: https://www.navcen.uscg.gov/pdf/AIS/SOLAS.V.19.2.1-5.pdf. Accessed Nov 26, 2019.

[7] International Telecommunication Union, *Technical Characteristics for an Automatic Identification System Using Time Division Multiple Access in the VHF Maritime Mobile Frequency Band*, International Telecommunication Union, Feb 2014.

[8] "Standard for interfacing marine electronic devices," Mobile, AL, Tech. Rep., 2002. Available: http://www.plaisance-pratique.com/IMG/pdf/NMEA0183-2.pdf

[9] Oasis of Seas. (2019). *Royal Caribbean Blog*. Regal Sea Media. [Online]. Available: https://www.royalcaribbeanblog.com/sites/default/files/articles/oasis-of-the-seas-bridge.jpg. Accessed: 2019-04-29.

[10] Computer Resource Center: Glossary. (n.d). National Institute for Standards and Technology. [Online]. Available: https://csrc.nist.gov/glossary/term/integrity. Accessed Mar. 20, 2019.

[11] Computer Resource Center: Glossary. (n.d). National Institute for Standards and Technology. [Online]. Available: https://csrc.nist.gov/glossary/term/authenticity. Accessed Mar. 20, 2019.

[12] M. F. Rubinstein, "Cybersecurity of rf apertures to maritime voyage networks," M.S. thesis, Dept. Electrical and Computer Engineering, Naval Postgraduate School, Monterey, California, Sep 2016.

[13] J. M. Heaney, "Transmitting arbitrary data with ais," M.S. thesis, Dept. Computer Science, Naval Postgraduate School, Monterey, California, Mar 2017.

[14] S. A. Marion, "Maritime rf security analysis," M.S. thesis, Dept. Electrical and Computer Engineering, Naval Postgraduate School, Monterey, California, Sep 2017.

[15] AIS frequently asked questions. (2019, Oct. 03). United States Coast Guard. [Online]. Available: https://www.navcen.uscg.gov/?pageName=AISFAQ. Accessed Mar. 13, 2019.

[16] AIS class comparison. (2019). U.S. Coast Guard. [Online]. Available: https://www.navcen.uscg.gov/pdf/AIS_Comparison_By_Class.pdf. Accessed May 30, 2019.

[17] A. Watson. (2016, Sep. 13). AIS - The Ultimate Guide to Automatic Identification Systems. Raymarine. [Online]. Available: http://www.raymarine.com/view/blog/news/details/?ID=15032385552. Accessed Jun. 26, 2019.

[18] AIS requirements. (2019, Aug. 14). United States Coast Guard. [Online]. Available: https://www.navcen.uscg.gov/?pageName=AISRequirementsRev#ClassB. Accessed May. 13, 2019.

[19] An explanation of AIS and Class B. (2008, Jan.). *Digital Yacht*. World Shipping Council. [Online]. Available: http://www.mesltd.co.uk/aisguide.pdf. Accessed May 28, 2019.

[20] AIS messages. (2019, Apr. 14). U.S. Coast Guard. [Online]. Available: https://www.navcen.uscg.gov/?pageName=AISMessages. Accessed Mar. 13, 2019.

[21] A. Goudossis and S. K. Katsikas, "Towards a secure automatic identitifcation system (ais)," in *Journal of Marine Science and Technology*, May 2018. Available: https://doi.org/10.1007/s00773-018-0561-3

[22] Combined Maritime Forces: CTF-151 counter piracy. (2019). Combined Maritime Forces. [Online]. Available: https://combinedmaritimeforces.com/ctf-151-counter-piracy/. Accessed Jul. 01, 2019.

[23] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. New York, NY: John Wiley Sons, Inc, 1996.

[24] International Maritime Organization. (2010, Jun.). Guidance for AIS Application Specific Messages. International Maritime Organization. London, UK. [Online]. Available: https://portal.helcom.fi/Archive/Shared%20Documents/AIS%20EWG%2021-2010_5-1-Inf%20AIS%20application%20specific%20messages.pdf. Accessed Jul. 01, 2019.

[25] A. Das and C. V. Madhavan, *Public-key Cryptography: Theory and Practice*, 1st ed. Boston, MA: Pearson, 2009.

[26] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, 1st ed. New York, NY: Springer, 2010.

[27] T. Cooper. (2020, Mar. 30). Fiber-optic Internet in the USA. BroadbandNow. [Online]. Available: https://broadbandnow.com/Fiber

[28] S. Mambra. (2019, Oct. 11). The Strait Of Dover —The Busiest Shipping Route In The World. Marine Insight. [Online]. Available: https://www.marineinsight.com/marine-navigation/the-strait-of-dover-the-busiest-shipping-route-in-the-world/

[29] E. Barker, "Nist special publication 800-57 part 1 rev. 4 recommendation for key management – part 1: General," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep., Jan. 2016. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

[30] P. Gutmann, "Pki: It's not dead, just resting," *Computer*, vol. 35, no. 8, pp. 41–49, Aug 2002.

[31] B. Kaliski, *RSA Digital Signature Scheme*. Boston, MA: Springer US, 2011, pp. 1061–1064. Available: https://doi.org/10.1007/978-1-4419-5906-5_432

[32] D. Hankerson and A. Menezes, *Elliptic Curve Signature Schemes*. Boston, MA: Springer US, 2011, pp. 407–408. Available: https://doi.org/10.1007/978-1-4419-5906-5_251

[33] D. Boneh, *Digital Signature Standard*. Boston, MA: Springer US, 2011, pp. 347–347.

[34] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proceedings of CRYPTO 84 on Advances in Cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 47–53. Available: http://dl.acm.org/citation.cfm?id=19478.19483

[35] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology* (ASIACRYPT '01). Berlin, Heidelberg: Springer-Verlag, 2001, pp. 514–532. Available: http://dl.acm.org/citation.cfm?id=647097.717005

[36] A. Boldyreva, V. Goyal, and V. Kumar, "Identity-based encryption with efficient revocation," in *Proceedings of the 15th ACM Conference on Computer and Communications Security* (CCS '08). New York, NY, USA: Association for Computing Machinery, 2008, p. 417–426. Available: https://doi.org/10.1145/1455770.1455823

[37] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Advances in Cryptology - ASIACRYPT 2003*, C.-S. Laih, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 452–473.

[38] Python package index. (2019). Python Software Foundation. [Online]. Available: https://pypi.org/project/charm-crypto/. Accessed Nov. 13, 2019.

[39] D. Moody, R. Peralta, R. Perlner, A. Regenscheid, A. Roginsky, and L. Chen, "Report on pairing-based cryptography," *Journal of Research of the National Institute of Standards and Technology*, vol. 120, p. 11–27, Feb 2015. Available: https://nvlpubs.nist.gov/nistpubs/jres/120/jres.120.002.pdf

[40] A. Saxena, "Extending the bls scheme to identity based signatures," 06 2006. Available: https://www.researchgate.net/publication/334001157_Extending_the_BLS_Scheme_to_Identity_Based_Signatures

[41] J. C. Cha and J. H. Cheon, "An identity-based signature from gap diffie-hellman groups," in *Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography: Public Key Cryptography* (PKC '03). London, UK, UK: Springer-Verlag, 2003, pp. 18–30. Available: http://dl.acm.org/citation.cfm?id=648120.746918

[42] J. Camenisch, S. Hohenberger, and M. O. Pedersen, "Batch verification of short signatures," in *Proceedings of the 26th Annual International Conference on Advances in Cryptology* (EUROCRYPT '07). Berlin, Heidelberg: Springer-Verlag, 2007, pp. 246–263. Available: http://dx.doi.org/10.1007/978-3-540-72540-4_14

[43] J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *Advances in Cryptology – CRYPTO 2004*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 56–72.

[44] F. Hess, “Efficient identity based signature schemes based on pairings,” in *Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography* (SAC ’02). London, UK, UK: Springer-Verlag, 2003, pp. 310–324. Available: http://dl.acm.org/citation.cfm?id=646558.694902

[45] B. Waters, “Efficient identity-based encryption without random oracles,” in *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques* (EUROCRYPT’05). Berlin, Heidelberg: Springer-Verlag, 2005, pp. 114–127. Available: http://dx.doi.org/10.1007/11426639_7

[46] C. Gentry, “Practical identity-based encryption without random oracles,” in *Advances in Cryptology - EUROCRYPT 2006*, S. Vaudenay, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 445–464.

[47] M. T. A. Products and Solutions. (n.d). AIS ABM BBM Message Encoder. Maritec Trust. [Online]. Available: https://www.maritec.co.za/aisabmbbmencode.php. Accessed Aug. 01, 2019.

[48] M. Agarwal. (2019, 06). What is the Speed of a Ship at Sea? [Online]. Available: https://www.marineinsight.com/guidelines/speed-of-a-ship-at-sea/. Accessed: 2019-11-13.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California