AUTOBAN: Secure Low-Power Body Area Network for Real-Time Physiological Status Monitoring

by

John H. Holliman III
B.S. in Computer Science and Engineering
Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2016

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2016

Certified by .....
Dr. Roger I. Khazan
Associate Group Leader, Secure Resilient Systems and Technology Group, MIT Lincoln Laboratory
Thesis Supervisor

Certified by .....
Michael A. Zhivich
Technical Staff, Secure Resilient Systems and Technology Group, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

AUTOBAN: Secure Low-Power Body Area Network for Real-Time Physiological Status Monitoring

by

John H. Holliman III

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Real-time monitoring of physiological data can reduce the likelihood of injury in noncombat military personnel and first-responders. MIT Lincoln Laboratory is developing a tactical Real-Time Physiological Status Monitoring (RT-PSM) system architecture and reference implementation named OBAN (Open Body Area Network), the purpose of which is to provide an open, government-owned framework for integrating multiple wearable sensors and applications. The OBAN implementation accepts data from various sensors enabling calculation of physiological strain information which may be used by squad leaders or medics to assess the team's health and enhance safety and effectiveness of mission execution. Security in terms of measurement integrity, confidentiality, and authenticity is an area of interest because OBAN system components exchange sensitive data in contested environments. In this thesis, I analyze potential cyber-security threats and their associated risks to a generalized version of the OBAN architecture. Using the threat analysis, I identify security requirements for RT-PSM systems and describe the development of a secure RT-PSM system, called the Authenticated and Trustworthy Open Body Area Network (AUTOBAN) proof-of-concept implementation, that meets those requirements using cryptographic primitives that operate efficiently on low-power embedded devices. The threat analysis and proof-of-concept application, are intended to inform the development of secure RT-PSM architectures and implementations.

Thesis Supervisor: Dr. Roger I. Khazan
Title: Associate Group Leader, Secure Resilient Systems and Technology Group, MIT Lincoln Laboratory

Thesis Supervisor: Michael A. Zhivich
Title: Technical Staff, Secure Resilient Systems and Technology Group, MIT Lincoln Laboratory

Acknowledgments

I would like to start by thanking Dr. Roger Khazan for allowing me the opportunity of being part of the Secure, Resilient Systems and Technology Group at MIT Lincoln Laboratory and making me feel welcome.

Thanks to Benjamin Fuller and Mayank Varia for providing recommendations and much needed information about securing embedded-device systems.

Thanks to Konstantin Feldman for reviewing and pointing out issues in my code.

Thanks to the people in Lincoln's Bioengineering Systems and Technologies Group, Brian Telfer, Albert Swiston, and others, for helping to orient me in the physiological-status monitoring space. I would also like to express gratitude towards Catherine Cabrera for overseeing this work and making it possible.

Lastly, I would like to give a special thanks to Michael Zhivich, who has helped me from the moment I started with the group, providing a sounding board for ideas, keeping the project on track, and especially advising and editing my thesis writing. This thesis would not have been where it is today without his help and guidance.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

The development of very low-power computing devices and wireless transceivers has improved the landscape of physiological-status monitoring by enabling creation of cheap, long-lasting devices that form a body area network (BAN). A BAN is a wireless network composed of wearable computing devices. BANs are typically seen in medical and personal fitness settings, but also have tactical military applications.

Both combat and noncombat military personnel endure tough environmental conditions and physical stresses that can lead to a variety of health issues, such as hypothermia, hyperthermia, musculo-skeletal injuries, hypoxia, and dehydration. It is well documented within the armed forces that the monitoring of real-time physiological data through an RT-PSM system can aid in anticipating the onset of these conditions, improving soldier health and readiness during training and in the field [43, 20]. For instance, members of a chemical, biological, radiological and nuclear defense (CBRND) team, who are often required to wear fully-encapsulating personal protective equipment, face significant risk of heat exhaustion which could be mitigated by providing CBRND team leaders a means of monitoring their team members' core body temperatures to properly coordinate actions [55]. In addition to improving the health and resilience of soldiers, an RT-PSM system would also increase squad leaders' situational awareness and provide actionable intelligence for mission planning.

The need for security in the BANs specifically is receiving ever more attention in the media and academic communities, especially with respect to medical devices

where the integrity and the availability of a system can be a matter of life and death. Diabetic Jay Radcliffe, for instance, demonstrated a threat to his health by compromising the wireless channel of his insulin pump and shutting it down [45].

The consequences of targeted adversarial attacks on BANs and tactical RT-PSM systems can be severe. An adversary who is able to create valid messages on a BAN's wireless channel will cause data to be unreliable, impacting both the integrity and availability of the system and potentially the success of the mission. A tactical RT-PSM system would necessarily exchange sensitive health information and data relevant to mission planning in a battlefield setting. Depending on the implementation, the system components may even interconnect to a tactical radio network. An attack on the RT-PSM system should not be able to impact other systems. Despite the multiple dimensions of usefulness, if the proper security mechanisms are not integrated into the design, tactical RT-PSM systems could allow for more harm than good.

In this thesis, I present an analysis of threats to a generalization on an existing tactical RT-PSM system architecture called OBAN. I provide a prioritized list of threats and define a corresponding set of requirements for enhancing security and privacy of RT-PSM systems. The analysis and threat ranking guide the development of a proof-of-concept RT-PSM system that balances security, privacy, and utility called the Authenticated and Trustworthy Open Body Area Network (AUTOBAN) proof-of-concept implementation. While this work was performed in the context of an RT-PSM system, many of the results generalize to the broader BAN and Internet of Things (IoT) spaces where sensitive information is collected and transmitted by low-power, computationally-limited devices.

The rest of the thesis is organized as follows. I provide relevant definitions and assumptions about the RT-PSM system architecture in Chapter 2. In Chapter 3, I present the risk analysis methodology and analyze threats to the system. I discuss a path forward to address the threats identified in the analysis, present work related to BAN security, and select appropriate cryptographic building blocks for a secure RT-PSM system in Chapter 4. Next in Chapter 5, I describe a proof-of-concept

RT-PSM system constructed using the selected building blocks and explain how it successfully thwarts a subset of the threats identified in Chapter 3. In Chapter 6, I apply a common framework for measuring the performance of cryptographic primitives on embedded devices to benchmark the selected building blocks used in the proof-of-concept application. Next, I detail the techniques I employed to ensure the cryptographic building blocks were implemented correctly, efficiently, and securely. Lastly in Chapter 8, I give concluding remarks and identify directions for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Real-Time Physiological Status Monitoring System Fundamentals

In this chapter, I define a generalized reference architecture and use cases based on OBAN RT-PSM system [19] for the threat analysis in order to ensure that the resulting recommendations are broadly applicable to a range of tactical RT-PSM systems. The primary aim of the OBAN system is to improve the health, preparedness, and overall resilience of noncombat military personnel through real-time monitoring of physiological data with devices capable of sustaining data collection and transmission for the duration of a mission.

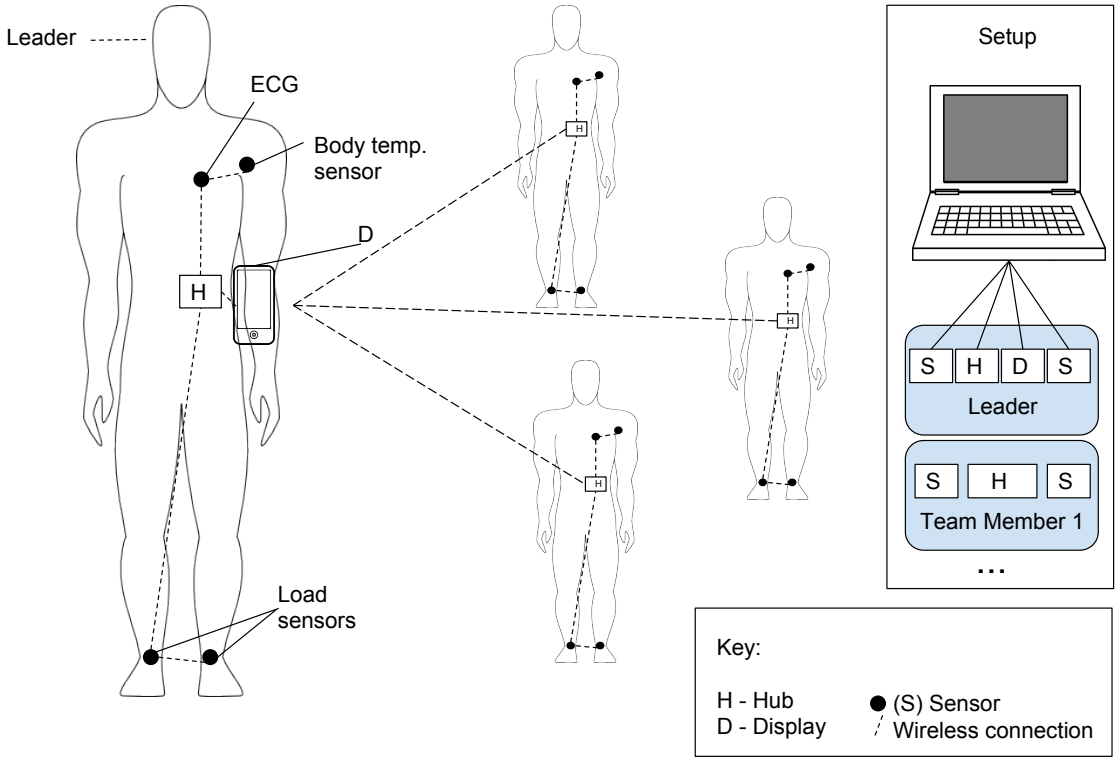
2.1 Core Components

The reference RT-PSM architecture consists of the following core components:

- *Sensors* — Data collecting devices (e.g., electrocardiogram (ECG), core temperature, load)
- *Hub* — A radio-enabled device that aggregates data from connected sensors
- *Display or end user device (EUD)* — A device that provides feedback to the user (e.g., a squad leader or a combat medic).

Hubs and *sensors* are low-power, resource-constrained devices; the OBAN prototype uses boards with an ATmega2560 microcontroller (16 MHz, 8-bit processor,

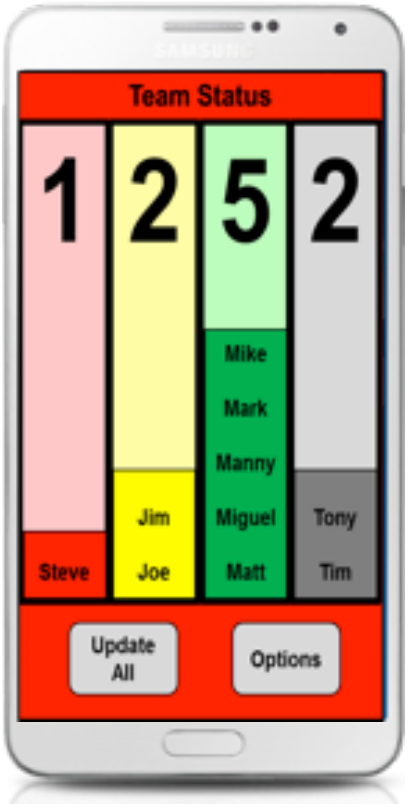
Figure 2-1: Example tactical RT-PSM system overview and configuration



256 KB program space, 8 KB memory) and a tunable narrow-band radio for sending data to the EUD [19]. Additionally, the hubs have microSD cards to store collected sensor data for post-mission analysis. *EUDs* are Android-based smartphones and thus are less resource-constrained than hubs; EUDs include custom radio dongles to communicate with hubs and an application to display RT-PSM data.

Team components are provisioned and configured to work together via USB connections to a configuration PC. At the beginning of a mission, multiple hubs are paired up with one or more display devices. Each squad member is outfitted with a hub attached to sensors. The display devices are carried by the squad leader and any medics. Figure 2-1 shows an example system setup.

Figure 2-2: Example EUD display, showing status of squad members.



2.2 Use Cases

For this analysis, I consider a use case that is targeted at supporting monitoring of a small team for the duration of a mission (3-7 days). Members of the team each have a hub device connected to several on-body sensors. The team leader and medic each have a display device (EUD) that gathers data from the hubs and uses that data to generate a summary of the team's status. Figure 2-2 shows screenshot of the OBAN team summary application running on an EUD.

The system uses an application-level messaging protocol with distinct message formats intended to support the target use cases. During missions, broadcast is the fundamental communication primitive. No trust assumptions are placed on the wireless communication infrastructure. Adversaries can take advantage of its broadcast nature to eavesdrop, destroy, or replay old messages. Messages do contain a cyclic redundancy check (CRC) to protect against accidental data modification.

During a mission, the system allows for a three modes of operation:

Status Updates (pull). To monitor the physiological strain of specific team members, the squad leader would use the pull-style mode of operation. During the mission, on the squad leader's prompting or at specified intervals, the display device would request data—in this case the Physiological Strain Index (PSI)—from the nearby hubs. The hubs expose PSI as a virtual sensor that is calculated from heart rate. Any hub close enough to receive the request will respond. This is the most common mode of operation.

Notifications (push). Squad leader and medic are interested in knowing if a squad member is entering a dangerous state that requires immediate attention. In these cases, a hub device must push information instead of waiting for it to be collected by query from an EUD. This mode of operation would be activated, for example, if a squad member's core temperature is in a dangerously high range, such as seen with heat exhaustion or stroke. In the push-style mode of operation, a hub sends a notification addressed to a particular EUD or group of EUDs. When the message is received, the EUD acknowledges receipt provided that acknowledgement was requested in the notification. A hub will continue resending a notification periodically until the message is acknowledged or it sends the notification a set threshold number of times.

Streaming. The streaming mode of operation is required if a medic is providing medical attention to a soldier and would like access to the soldier's health data (e.g. ECG feed). In the stream-style mode of operation, an EUD requests that a hub send (stream) batches of a certain type of data for a specified amount of time. The hub responds with batches of data until the request has been fulfilled. Dropped batches are ignored. If the EUD wants to maintain the incoming stream it will need to send a new stream request message after the hub completes the previous stream request. This is the least common mode of operation as transmitting is power intensive [44].

Chapter 3

Threat Analysis

This threat analysis considers a range of potential adversaries, their capabilities and methods of attack, and presents a qualitative measure of the risks from such attacks, culminating in a prioritized list of threats. While this threat analysis should be considered qualitative, not quantitative, I endeavor to offer an approach that can be used for a variety of adversaries, tactics, or system configurations in the RT-PSM space.

Many of the security concerns that BANs face have been identified in [57] and [50]; while both of these papers are not specific to tactical RT-PSM systems, they offer detailed exploration of security concerns in the context of resource-constrained, RF-enabled devices. Additional threats that challenge the security of the tactical RT-PSM system, including threats to the communication interface, the device software, and the device hardware, are identified.

3.1 Analysis Methodology

To analyze the risk presented by a particular threat I use a qualitative method based on an approach detailed in [49], which recommends using a standard model where *risk* is a function of *impact* and *likelihood*. Risk is computed as the product of impact and likelihood, where impact is a subjective measure of the magnitude of harm that can be expected to result from a successful attack, and likelihood is a subjective measure

of an adversary's ability to obtain requisite capabilities. Impact to the reference RT-PSM system security is computed as the maximum loss of confidentiality, data or source integrity, or availability due to the attack. Each of these quantities is separately estimated on a scale from 0 (no loss) to 5 (largest loss possible). To estimate threat likelihood, I consider an adversary's ability to obtain the requisite capabilities, his motivation to perform an attack, and its alignment with adversary's goals.

3.2 Adversarial Model

Following standard computer security practice, I consider adversaries that differ from one another in goals, capabilities, and roles within the system. Specific adversaries for this threat analysis are listed in Table 3.1.

Goals: Adversaries' goals affect their motivation and choice of attacks. For instance, a squad member may not wish to participate in a particular mission and may manipulate the system to that end (e.g. fake sensor readings). A nation state, alternatively, might be interested in increasing its situational awareness by determining presence of troops and acquiring the associated physiological data. A nation state adversary would thus mount an attack very different in stealth and scale from one employed by a squad member. Table 3.1 lists example goals for each adversary considered in this analysis.

Roles: An adversary may be an *outsider* or an *insider* with respect to the system. An insider is someone who has a legitimate role in the system's operation, development, or maintenance (e.g., a user, a manufacturer, a system administrator). An outsider is an entity that comes or puts itself in contact with the system (e.g., civilians near troops with an RT-PSM deployment, an opposing military force, etc). The level of access provided by a particular role plays a part in what attacks an adversary can successfully execute.

Capabilities: Mounting attacks requires a set of capabilities, which might include the ability to receive and send RF communications, knowledge of the RT-PSM message protocol, etc. Capabilities are a function of available resources; to describe

Table 3.1: Example adversaries with associated goals and estimated capability level

	Adversary	Example Goal	Capabilities
Outsiders	Script kiddie	Create personal amusement by bragging about system disruption	Tier I
	Hacktivist	Disrupt the system or publicize sensitive information for political gain	Tiers II-III
	Nation state	Decrease the situational awareness of RT-PSM system users	Tiers V-VI
Insiders	Squad member	Malingering	Tiers I-II
	Squad leader	Improve unit appearance by faking physiological data	Tiers I-II
	System administrator	Profit from health information extracted from the system	Tiers III-IV
	Hardware/Software supplier	Profit from health information extracted from the system	Tiers IV-V

adversary capabilities I adopt the hierarchy presented in [8], which describes six tiers of adversaries in order of increasing sophistication: Tier I-II attackers use exploits written by others, Tier III-IV attackers can write their own exploits and discover new vulnerabilities, and Tier V-VI attackers are well-funded and able to implant vulnerabilities.

The following specific capabilities are necessary to stage attacks against the RT-PSM system:

- *Receive RF* — The ability to receive transmissions on the appropriate frequencies.
- *Send RF* — The ability to send transmissions on the appropriate frequencies.
- *Receive Message* — Decipher an RF transmission into a message through knowledge of the application-level messaging protocol.
- *Send Message* — Construct a valid message through knowledge of the application-level messaging protocol.
- *Hub HW/SW Mod.* — Ability to modify hardware or software on the hub

Table 3.2: RT-PSM system adversaries and likelihood of obtaining capabilities

Adversary	**Tier**	**Capabilities**							
		Receive RF	Send RF	Receive Message	Send Message	Hub HW/SW Mod.	EUD HW/SW Mod.	Hub Capture	EUD Capture
Script kiddie	I	0.1	0.2	0.1	0.2	-	-	-	-
Hacktivist	II-III	0.8	0.8	0.8	0.8	-	-	0.1	0.1
Nation state	V-VI	1	1	1	1	1	1	0.8	0.8
Squad member	I-II	0.2	0.2	0.2	0.2	0.8	0.4	0.4	0.4
Squad leader	I-II	0.2	0.2	0.2	0.2	0.4	0.8	0.6	0.6
System admin	III-IV	-	-	-	-	0.8	0.8	0.8	0.8
HW/SW supplier	IV-V	-	-	-	-	1	1	-	-

devices.

- *EUD HW/SW Mod.* — Ability to modify hardware or software on the EUD devices.
- *Hub Capture* — Obtain a fielded hub device.
- *EUD Capture* — Obtain a fielded EUD device.

Informed by these capabilities, I describe the adversary models below; please refer to Table 3.2 for a summary.

The model for a *Script Kiddie* adversary implies someone with little sophistication but with access to easily purchasable hardware and downloadable tools. Since the radio components used in the reference RT-PSM system are readily available, and software for parsing message data is not difficult to obtain, I believe this adversary can obtain capabilities to send and receive messages. However, putting the whole system together requires some effort and the motivation for this adversary is fairly low, thus the overall likelihood is rated at 20%.

A *Hacktivist* adversary is similar in nature to a *script kiddie*. However, hacktivists differ in having a much stronger motive, which may embolden them (though with low

probability) to capture a hub or EUD device. Since a hacktivist's motivation is higher, the corresponding likelihood of obtaining capability to send and receive messages is rated higher as well.

The model for a *Nation State* adversary is an entity that is able to obtain any of the required capabilities. Capturing hub or EUD devices may prove more difficult, so these likelihood values are below 100%.

A *Squad Member* adversary is a legitimate user of the system, and thus has access to the hardware and software running on both hubs and EUDs. This adversary's ability to put together a system from readily-available components may be higher than that of a *script kiddie*, but the motivation is lower, so the ranking for sending/receiving messages (independent of the device they already have) remains at 20%. Due to more access to the system, this adversary has a higher likelihood of modifying a hub or an EUD, or possibly obtaining one from another squad member or leader.

A *Leader/Medic* adversary is similar to the *squad member* model; however, this adversary has greater access to EUDs, thus the corresponding change in likelihood assignments.

A *System Admin* adversary is someone who is in charge of provisioning the system for operation and thus has ample opportunity to modify hardware or software on hubs or EUDs, as well as obtain one of these devices after field deployment.

A *Hardware/Software Supplier* adversary, of course, has ample opportunity to insert malicious trojans into the hardware or software of the device they are creating.

Table 3.2 presents my estimates of the likelihood that adversaries obtain the capabilities necessary to mount attacks against the reference RT-PSM system. While specific values in this table are certainly subjective, the relationship between adversaries and the capabilities they can obtain corresponds to the descriptions in the Defense Science Board report [8].

Table 3.3: RT-PSM system threats, required capabilities, and threat impact.

	Attack ID/Description	**Capabilities Required**								**Loss of...**			
		Receive RF	Send RF	Receive Message	Send Message	Hub HW/SW Mod.	EUD HW/SW Mod.	Hub Capture	EUD Capture	Confidentiality	Data Integrity	Source Integrity	Availability
Passive	P1. Observe presence of communication on RF frequencies used by RT-PSM system	✓								1	-	-	-
	P2. Observe communication patterns (e.g. query/response) to identify squad leader	✓								2	-	-	-
	P3. Observe unique identifiers of devices in communication contents	✓		✓						3	-	-	-
	P4. Learn sensor data from communication contents	✓		✓						4	-	-	-
Active	A1. Spoof query message to trigger a response; detect/locate responding devices	✓	✓		✓					2	-	2	2
	A2. Spoof query message to trigger a response; parse response for unique ID	✓	✓	✓	✓					3	-	2	2
	A3. Spoof query message to trigger a response; parse response for sensor data	✓	✓	✓	✓					5	-	2	2
	A4. Send fake response data from a particular device	✓	✓	✓	✓					4	-	2	2
	A5. Send “wake up” preamble repeatedly		✓							-	-	-	4
	A6. Send garbage messages to prevent others from receiving successfully		✓							-	-	-	5
Full-Scope	F1. Modify HW/SW on hub to broadcast sensor data as desired	✓		✓		✓				3	-	-	2
	F2. Modify HW/SW on hub to send incorrect message data in response to queries					✓				-	3	3	2
	F3. Modify HW/SW on EUD to leak all collected sensor data						✓			5	-	-	1
	F4. Modify HW/SW on EUD to store or display incorrect data to leader/medic						✓			-	5	5	-
	F5. Learn sensor data for an individual stored on the SD card							✓		3	-	-	-
	F6. Learn sensor data for a group stored on the SD card								✓	5	-	-	-

3.3 Threats and Impact

In this section, I present threats to the reference RT-PSM system and rank them using the methodology described in Section 3.1. Table 3.3 describes the attacks on the reference RT-PSM system and the capabilities required to mount these attacks; it also shows how these attacks affect the security properties that inform the impact metrics, as described below.

Specific attacks fall into three distinct classes: *passive* attacks that use data sent over the wireless communications channel, *network active* attacks that involve reading, creating, or destroying transmissions on any communications channel, and *full-scope* attacks that are not limited to the communications channel and may include software or hardware modification, device capture, etc. The attack class relates both to necessary capabilities and the resulting impact on system security.

The *Confidentiality* column shows to what extent attacks are successful at exfiltrating data from the RT-PSM system. The majority of attacks target identifying or health-related information; consequently, most entries in this column are non-zero. Attacks that leak larger quantities of data or more sensitive data receive higher impact ratings.

Non-zero entries in the *Data Integrity* column correspond to attacks that enable undetected modification of data. Note that certain attacks that may result in message corruption (e.g. denial of service) are not in this group, since these issues are detected by checking the message's CRC checksum. On the other hand, attacks such as *F1. Individual privacy compromise* and *F4. Group data or source spoofing*, that rely on hardware/software modifications cannot be detected using similar techniques.

The *Source Integrity* column shows attacks that require or involve impersonation of certain entities. The reference RT-PSM implementation assumes that communicating entities are legitimate and use unaltered system components. The non-zero entries correspond to attacks that take advantage of this naive assumption.

The *Availability* column shows how much attacks disrupt the reference RT-PSM system. Active attacks such as *A2. Privacy compromise (on demand)* are shown as

Table 3.4: Top RT-PSM threats based on estimated system risk

Attack	**Adversary**	**Risk (0-5)**
A3. Privacy compromise (on demand)	Nation state	5
F3. Group privacy compromise	Nation state HW/SW Supplier	5
F4. Group data or source spoofing	Nation state HW/SW Supplier	5
A6. Jamming	Nation state	5
P4. Privacy compromise (opportunistic)	Nation state	4
A3. Privacy compromise (on demand)	Hacktivists	4
F3. Group privacy compromise	Leader/Medic System Admin	4
F6. Group privacy compromise (post hoc)	Nation state System Admin	4
A4. Identity spoofing	Nation state	4
F4. Group data or source spoofing	Leader/Medic System Admin	4
A5. Battery drain	Nation state	4
A6. Jamming	Hacktivists	4
P4. Privacy compromise	Hacktivists	3
A4. Identity spoofing	Hacktivists	3
A5. Battery drain	Hacktivists	3

having an effect on availability because they cause the system components to transmit more frequently, reducing battery life; furthermore, such attacks do not respect the medium access control scheme and might interfere with legitimate message traffic.

3.4 Top Ranked Threats

Table 3.4 presents the top-ranked threats against the reference RT-PSM system for the adversary models. This ranking was obtained by computing risk as a product of attack's impact to the system (maximum value across impact columns of Table 3.3) and attack's likelihood (minimum value across required capabilities in Table 3.2). The maximum impact is selected to highlight where the attack is most damaging;

this assumes loss of confidentiality, data integrity, source integrity, or availability are equally damaging. The minimum likelihood is selected for obtaining required capabilities since the likelihoods listed in Table 3.2 are often not independent (e.g. an adversary with *RF receive* capability has *RF send* capability as well), and the minimum likelihood represents a reasonable upper bound of the joint likelihood.

As expected, the more capable adversaries pose the most threat; however, it appears that hacktivist-level attacks can cause as much impact to the reference RT-PSM system (in certain categories) as those mounted by a nation state. Most of the top threats result in loss of confidentiality due to leakage of sensor data that represents the health information of RT-PSM users; however, the ways of achieving this privacy violation vary significantly. While loss of privacy is certainly undesirable, these threats do not by themselves render the system unusable.

Another group of top threats pertains to information integrity, enabling adversaries to spoof identity of sender or fake the data itself. Since integrity violations bring into question the trustworthiness of the system, they are more worrisome than privacy risks. For example, a leader or medic can be misled about a team member's health status resulting in incorrect decisions that risk the mission and the team's well-being.

Finally, the *jamming* and *battery drain* attacks also make an appearance. These attacks strike at the availability of the RT-PSM system to perform its function; since these attacks are relatively easy to mount, this risk brings the reliability of the resulting system into question.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Building a Secure Real-Time Physiological Status Monitoring System

In this chapter, I identify an appropriate strategy for mitigating the threats identified in Chapter 3 that will work in low-power devices without compromising the system's original goal of providing RT-PSM services to improve the health and resilience of military personnel and first-responders. First, I outline the security requirements for a RT-PSM system, based on the threat ranking in Section 3.3. Next, I describe relevant work related to providing security to resource-limited devices. Lastly, I select specific cryptographic primitives that best fulfill the security goals.

4.1 Requirements for a Secure Real-Time Physiological Status Monitoring System

Informed by top threats discussed in Section 3.3, I would like to reduce the risk resulting from cyber attacks to an acceptable level. In order to accomplish this, I need to decrease the likelihood of a threat (e.g., increase the difficultly of obtaining the necessary capabilities to mount the attack) or reduce the impact of the threat

(e.g., increase the reference RT-PSM system's ability to withstand a certain attack). To accomplish this goal, the following security requirements should be considered during design and implementation of an RT-PSM system.

Data integrity: Data in transit between a hub and an EUD should not be corrupted or modified without the ability for detection. This is required to mitigate identify spoofing attacks (*A1-A4*) by significantly increasing the cost of obtaining unauthorized Send Message capability.

Authentication and Authorization: Components should be able to verify that a particular entity (or group of entities) created a message. This is required to mitigate attacks that rely on spoofing identity of a system user (e.g. squad leader), who may have privileges that other members do not (e.g. requesting status updates). This capability would help mitigate attacks *A1-A4*, *F2*, and *F4*. Adding authentication and authorization to RT-PSM system would enable revocation of privileges from misbehaving devices, like those that result from attacks *F2* and *F4*.

Data confidentiality: Data should be readable only by authorized components of the RT-PSM system. Parties outside the BAN should not be able to read sensor data in transit or from the memory or storage of a captured device. The second requirement may be difficult to achieve given the need to store any cryptographic keys in device memory. Encrypting the data in transit will help mitigate attacks P3, P4, A1-A3; encrypting data at rest will mitigate attacks F5-F6.

Data freshness: Authentication and confidentiality are not enough to prevent harm from active attacks that replay valid messages. In order to prevent replay attacks, hubs and EUDs should be able to ascertain the freshness of received messages by including and verifying a timestamp, sequence number, or nonce in the message.

Software integrity checking: RT-PSM systems should use code signing or other methods of checking software integrity to ensure that software placed on hubs and EUDs has not been tampered with to leak data or provide incorrect information to the squad leader. Ensuring software integrity during configuration and execution will help mitigate attacks F1-F4 that rely on tampering with software. While hardware modifications may still be able to accomplish these attacks, they are typically much

more expensive to execute successfully.

Availability and Sustainability: Hubs and EUDs should always be able to send and receive RT-PSM data. All components of our system should support data collection for the duration of a 3 to 7 day mission. This requirement implies that cryptographic algorithms used to ensure other requirements should not be overly expensive in terms of required communication overhead or computational power.

These security requirements are unordered, since a precise ranking would be use case dependent. For instance, availability may not be as important to an application constantly streaming ECG data as it might be for an application notifying users of exposure to hazardous chemical agents. That said, these security properties are applicable to the majority of tactical RT-PSM systems.

4.2 Existing Techniques

In this section, I present related work to provide the necessary context for devising an RT-PSM system that achieves the requirements given in Section 4.1.

The Setting. Although much of the research around BANs and wireless sensor networks has been focused on making them possible and useful, an increasing number of researchers are investigating the problem of how to provide security services and trustworthiness to resource-constrained devices.

In two of the earlier papers on the subject [41, 44], the authors construct a secure wireless sensor network built on resource-constrained devices running TinyOS. The system aims to provide data confidentiality, data authentication, data integrity, and data freshness. The resource-constrained sensors communicate exclusively with a more powerful central base station. The authors evaluate several block ciphers, including AES, and settle on running a finely tuned RC5 [46] cipher in counter mode, due to the cipher's small code size and high speed. For message authentication, they use CBC-MAC [17], allowing them to reuse the same RC5 block cipher. They apply more code reuse by using the CBC-MAC algorithm for pseudo-random number gen-

eration. A lot of their design decisions are relevant to a tactical RT-PSM system, but they do not offer a complete solution for this use case. I adapt lessons learned from their system to the RT-PSM setting.

Much of the BAN research focuses on implantable and wearable devices in the context of modern medicine [50, 34, **?**, 31, 21]. In medicine, small embedded devices are responsible for relaying sensitive patient information or administering doses of medication. The need for security and privacy in this domain comes as a result of increased legislation around protecting patient information and understanding of the consequences of an insecure system. In spite of this, the hacking of medical devices has been (and currently is) far from the number-one risk to public health [31]. Security mechanisms create additional opportunity for bugs and can slow down regulatory approval [21]. Further, security mechanisms work directly against the already limited program space and power resources. As a result, typical BAN components have been designed to transmit unencrypted data on unauthenticated channels, allowing eavesdropping and identity spoofing attacks.

In order to build a trustworthy system, as articulated by Burleson et al., designers must aim to [21]:

- Consider security early in the development process (as it is difficult to retrofit)
- Encrypt sensitive information—at rest or in transit—whenever possible
- Develop a realistic threat model with realistic assumptions (e.g. adversary can obtain source code and design documents)
- Use industry-standard source-code analysis techniques and cryptographic building blocks

The last design goal proves especially difficult to apply in practice. Cryptographic building blocks are not available or standardized for use in the most resource-constrained devices. Medical device manufactures do not make their source code public and the research community has not produced any complete, widely-accepted solutions. Of course, there are plenty of authentication and key agreement building blocks and protocols for less resource-constrained environments, such as Transport Layer Security (TLS) or Elliptic Curve Diffie-Hellman. Unfortunately, the majority of these schemes

depend on cryptographic primitives that need computational power or code space at odds with that available for most embedded device systems [52].

Work focused on porting solutions from different computing environments is widespread in the literature. For instance, researchers have gone through great lengths to tune the AES block cipher for lightweight applications. The best known hardware implementation of AES-128 is down to 2400 gate equivalents [42]—an impressive accomplishment[1] but this is still too large and not compatible with the system components' other goals (e.g., low-current draw during sleep) [57, 15]. This motivates the development of lightweight cryptographic primitives and BAN-specific authentication techniques.

Lightweight Cryptographic Primitives. Recently, the U.S. National Security Agency (NSA) developed Simon and Speck—families of block ciphers designed for severely resource-constrained environments and optimized for hardware and software implementations respectively [13]. Simon and Speck require a fraction of the power of optimized AES cores for equivalent encryption strength [15].

BAN-Specific Authentication Techniques. The SoK literature study evaluates BAN-specific authentication techniques [50]. Some of note include:

- *Biometrics* — Using physiological values (e.g. ECG data) as a source of entropy for key generation. This allows devices to take advantage of the fact that they are situated on the same body. Unfortunately, biometrics does not help when devices are situated on different bodies.
- *Distance-bounding authentication* — Establishing physical distance between two communicating parties based on the timing delay between sent and received messages. This provides weak authentication, but works well if adversaries are not within a certain range.
- *Out-of-band authentication* — Utilizing channels like audio or visual for communication. Provides authentication only if the adversary will not be snooping

[1] To put this in context, according to [57] the amount of power it takes certain AES circuitry to encrypt a 29-byte packet is 10 times less than it takes a TI MSP430g2553 (an extremely low-power micro-controller) to turn on!

the auxiliary channel.

- *Anomaly detection* — Detecting changes in behavior—e.g. resource depletion rate, communication patterns, etc.–and responding appropriately.

Key Management. Cryptographic primitives require keys. In the OBAN system, devices are batch provisioned before a mission. This simplifies the issue of distributing keys, allowing for keys to be pre-placed on devices. Future tactical RT-PSM applications may not allow for pre-placement of keys. For these applications, lightweight, usable key management and key distribution solutions, such as the Lincoln Open Cryptographic Key Management Architecture [7], will be necessary.

Summary. One of the challenges in developing general purpose security mechanisms for BANs is that compute resources, power resources, and specific use cases differ drastically. Tactical RT-PSM system components fall somewhere between medical devices and fitness trackers. Medical devices are on the extreme end of power efficiency. A pacemaker, for instance, must last on several years on a non-rechargeable battery [35] and in the absolute worst case a compromise of the wireless channel could result in death. Fitness trackers on the other hand are typically recharged once per day and in the worst case a compromise of the wireless channel would result in a small privacy violation (see [12] for a comical illustration of this).

4.3 Selecting Security Building Blocks

This section details the security building blocks I selected to achieve the security requirements I established in Section 4.1.[2] Those requirements are:

- Data integrity
- Authentication and Authorization
- Data confidentiality
- Data freshness
- Availability and Sustainability

[2]Software integrity checking is not included due to time constraints and is left to future work.

Table 4.1: Options for providing data integrity

Security Goal	Strategies		
	(Unkeyed) Hash	MAC	Digital signature
Data Integrity (accidental)	✓	✓	✓
Data Integrity (intentional)	-	✓	✓
Authentication	-	✓	✓
Non-repudiation	-	-	✓
Entity separation	-	-	✓
Kind of keys required	None	Symmetric	Asymmetric

The first step in my process of selecting mechanisms to fulfill the requirements is to consider different ways of providing each requirement individually, which involves looking at standard techniques and the BAN-specific techniques discussed in Section 4.2. After coming up with various strategies for satisfying individual requirements, I discuss how to combine them into a single cohesive solution.

4.3.1 Data integrity

A strategy provides data integrity if it gives the system components the ability to detect accidental and intentional changes in the data. There are three main options for providing message data integrity (summarized in Table 4.1), hash, message authentication code (MAC), and digital signature, and they are distinguished by the security goals they fulfill. The primary interest is data integrity, but it is worth mentioning that some of the options can be used to meet other goals. There are two forms of data integrity of interest: detecting accidental changes in data and detecting intentional changes in data.

In the first option, hash, an unkeyed hash of the message is taken and appended to the message. This protects against accidental changes to the message. A hash will not protect against intentional data modification because an adversary can simply recalculate the hash after modifying the message and append it in place of the original[3]. The OBAN prototype uses an implementation of this scheme called cyclic

[3]Although, it is worth noting that real-time modification of RF-based communications is ex-

redundancy check (CRC). The CRC codes used in OBAN are 2 bytes.

The second option, message authentication code (MAC), provides both forms of data integrity and authentication (see Section 4.3.2). Authentication means that the message recipient can be confident that a message originated from a specific sender or group of senders. While a MAC is sometimes referred to as a *keyed hash*, it can be constructed from block cipher algorithms as well as cryptographic hash functions (e.g. keyed-hash message authentication code (HMAC) construction). In either case, the algorithm will accept a secret key and an arbitrary-length message and output a sequence of bits known as a *MAC* or *tag*. Both varieties of MAC algorithm can be implemented efficiently for resource-constrained devices. Tags are not long; typically (and depending on the implementation), the length of a tag is t bits where $t - 1$ is the security level[4] measured in bits (e.g., a 128-bit tag can offer 127 bits of security) [24].

The third and last option, digital signatures, provides both forms of data integrity, authentication, and non-repudiation. Non-repudiation is not a requirement for a secure RT-PSM system, but I mention it for completeness. Non-repudiation is the property that the message recipient can prove to a third party that a message originated from the sender. MACs do not provide this property because both the sender and receiver have the same secret key and a third party would be unable to determine whether or not the party claiming to be the recipient received the message or created it. Digital signatures require asymmetric keys, also known as *public-private key pairs* (described in [53]). Producing a signature involves hashing the message to generate a *digest* and using the private key to encrypt the digest. The signature is verified with the public key, but can only be created with the corresponding private key, allowing for entity separation—the property that an entity's role (e.g., message sending) is limited by the security method. One of the most efficient means of providing digital signatures is the NIST standard (FIPS 186-4) Elliptic Curve Digital Signature Algorithm (ECDSA) [39]. The length of an ECDSA signature is about $4t$

tremely difficult. That said, there is still the issue of an adversary being able to create messages.

[4]At a security level of 127 bits, for instance, an attacker would need 2^{127} operations to discover the secret key

Table 4.2: Options for providing authentication

Security Goal	Strategies					
	Digital Signatures	MAC	Username/password	Distance bounding	Out-of-band	Zero-knowledge proofs
Authentication	✓	✓	✓	✓	✓	✓
Entity separation	✓	-	✓	-	-	✓
Kind of keys required	Asymmetric	Symmetric	Password	None	None	Password

bits where t is the security level in bits. It is possible to implement digital signature algorithms for resource constrained devices, but it is difficult. Signature validation on microcontrollers, using a highly-optimized ECDSA implementation can take on the order of hundreds of milliseconds to tens of seconds [58]. This is not acceptable for the RT-PSM system use cases described in Chapter 2. Additionally, the public-key based method also requires a large amount of code space and memory usage, thus making the algorithm unsuitable for RT-PSM systems [52].

4.3.2 Authentication and Authorization

A strategy provides authentication if it allows the message recipient to confirm that the message originated from a particular sender or group of senders. Authorization refers to rules that determine what certain entities are allowed to do (e.g., only medics are allow to request an ECG stream). Authentication and authorization are closely related with authentication enabling authorization to occur. That is, authorization has to be layered on top of an authentication scheme. The strategies considered for providing authentication are given in Table 4.2.

The first two options, digital signatures and MACs, were discussed above with respect to data integrity.

The next option, username/password-based authentication, refers to the strategy

employed by many websites on today's internet. Websites using this strategy authenticate users by requiring a password[5]. The password is sent from the browser to the server over TLS-encrypted channel, hashed, and compared to a hash of the password stored by the server when the account was created. Not storing a secret on the device would help prevent issues that arise from a device being captured (attacks *F5-F6*). Unfortunately, in a tactical RT-PSM system, it is not possible to require users to enter passwords.

The next two options, distance bounding and out-of-band authentication, were identified in Section 4.2 and make use of BAN-specific features. Distance-bounding authentication works by determining the distance between two communicating parties based on the timing delay between sent and received messages. This method only works if the adversary is outside the range of the system's radios. Furthermore, it would require that the leader and hub devices to send multiple messages to authenticate themselves [50]. For example, in the pull-style mode of operation described in Chapter 2, the leader would request data from the squad member, the squad member would ask the leader to repeat something[6], and the leader would have to respond before the squad member will reply with the desired data. This is undesirable because it would necessitate increased transmissions, which require power, and the development of a more robust medium access control[7]. Out-of-band authentication works by utilizing channels like audio or visual for communication and is successful if the adversary will not be snooping the auxiliary channel [50]. This method is undesirable because it requires additional hardware for accessing the auxiliary channel and potential user interaction.

The last option, zero-knowledge proofs, or more specifically zero-knowledge password proofs (ZKPP), provide a message sender the ability to prove to a message receiver that they know a password without revealing any information other than

[5]And, the browser authenticates the website using Transport Layer Security (TLS) or Secure Sockets Layer (SSL) which are based on public-key cryptography

[6]Perhaps a random value, so that a far away adversary will not be able to time messages in such a way so as to appear nearby.

[7]The OBAN reference implementation relies on a simple time division multiplexing scheme that would not support distance-bounding authentication without modification.

the fact that they know the password to the message receiver. The ZKPP algorithm is described in the IEEE P1363.2 draft [4] and is considered part of the Password Authentication Key Exchange (PAKE) family of protocols. As with distance bounding, PAKE protocols require multiple messages and are very expensive. Optimized implementations, for instance, have been shown to take hundreds of milliseconds to run on Android smart phones [30].

4.3.3 Data confidentiality

Data confidentiality protects data from unauthorized access. That is, with a data confidentiality mechanism in an RT-PSM system unintended message recipients, such as attackers, will not be able to discern message contents.

There are two main options for providing data confidentiality: 1) public-key based encryption and 2) symmetric-key based encryption.

In public-key based encryption schemes, each party has both a widely distributed public key and a secret private key only know to the owner. Any message sender in this system can encrypt a message with the recipient's public key, but the message can only be decrypted with the recipient's private key. As suggested above in the discussion of digital signatures, public-key based encryption schemes are not appropriate for resource constrained devices.

In symmetric-key based encryption schemes, the same cryptographic keys are used for both encryption and decryption. Symmetric-key encryption schemes can use stream cipher or block cipher algorithms. In stream ciphers, plaintext is encrypted one bit at a time using a key stream generated from the cipher. In block ciphers, plaintext is encrypted in fixed-size blocks. For more information on stream ciphers, please refer to the RSA technical report on stream cipher operation [47]. For more information on block ciphers, consult Chapter 3 of Mihir Bellare's and Phillip Rogaway's "Introduction to Modern Cryptography" [18]. Both stream and block ciphers can be implemented efficiently for low-end hardware. An important difference between the two is that stream ciphers are not capable of being combined with a cipher mode of operation that provides integrity protection or authentication.

There are also hybrid methods, such as Elliptic Curve Diffie-Hellman [28], which use public-key methods to establish a symmetric key for further communication. If key distribution in the field is required, hybrid methods might be necessary. Further work is needed to determine their feasibility because as with ECDSA they do not appear to be appropriate for resource-constrained devices[8].

4.3.4 Data freshness

Data freshness provides message recipients the ability to determine if a message has been received before which is necessary to prevent an adversary from recording and then successfully replaying a message that would otherwise include a correct signature or MAC. Nonces can be used to provide freshness. A nonce can be a random value, a timestamp, or a counter. A message sender includes a nonce in the message and the receiver remembers past nonces. If the receiver detects a message with a nonce it has seen before, it discards the message. Unlike random values and counters, timestamps would allow a receiver to determine that a message was sent recently. Although, clock skew between devices may limit the effectiveness of timestamps. Nonces must be combined with a data integrity mechanism, otherwise an adversary could simply alter the nonce.

4.3.5 Availability and Sustainability

Any mechanisms for providing the security requirements detailed so far must be able to run on the leader device and the hub devices so as to allow for data collection and system operation to continue for the duration of a mission. The hub devices are the limiting factor because they are the most resource constrained components in terms of power, memory, and computational ability.

A tactical RT-PSM should be functional as much as possible over the course of a mission. Chapter 3 described how active attacks, such as *A2, Privacy compromise (on demand)*, can affect availability because they cause the system components to

[8]The rate at which keys are distributed/changed is a determining factor here.

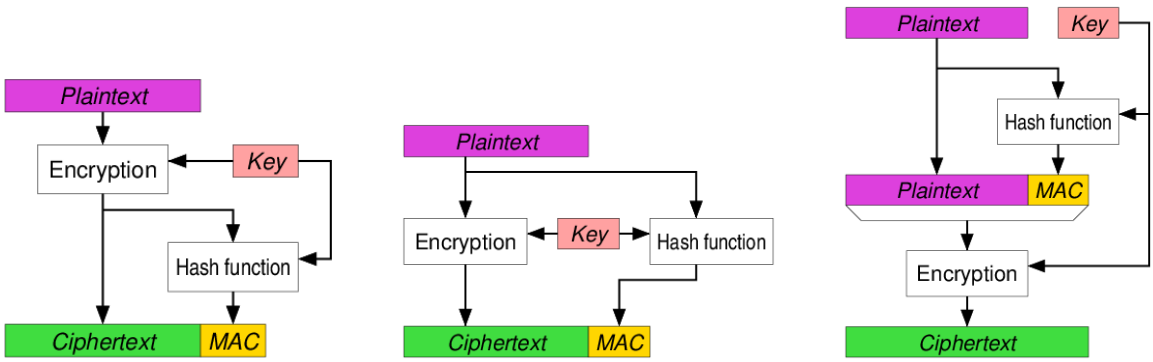
transmit more frequently, reducing battery life. In order to preserve availability, the system must be able to filter out illegitimate message traffic quickly and efficiently. Data integrity, authentication, and data freshness can be combined to detect illegitimate traffic. Anomaly detection can be layered on top of these mechanisms to further reduce battery loss from active attacks. That is, a device could shut off its radio if too many invalid messages have been detected in a certain amount of time. This method trades availability in the present for availability in the future.

4.3.6 Forming a cohesive solution

Based on the discussion above, it appears that the strategies for providing data freshness and availability are dependent on data integrity, authentication, and data confidentiality. It is also apparent that cryptographic primitives are needed in order to provide data integrity, authentication, and confidentiality. Symmetric cryptographic primitives are the best candidates for low-cost RT-PSM implementations.

In order to provide data integrity, authentication, and confidentiality, the solution must combine a symmetric-key based encryption mode with a MAC. This is referred to as an Authenticated Encryption (AE) mode or Authenticated Encryption with Associated Data (AEAD) mode. The difference between an AE mode and an AEAD mode is that in an AEAD mode some of the data is covered by the MAC but not encrypted. For an RF-based messaging system, an AEAD mode is more appropriate. In order to achieve sustainability, it is important that the system does not encrypt message headers to allow devices to filter out legitimate messages not intended for them as quickly as possible.

In the NIST publication on EAX, an AEAD mode of operation, when motivating the need for EAX, Bellare et al. note, "that people had been doing rather poorly when they tried to glue together a traditional (privacy-only) encryption scheme and a message authentication code (MAC)" [16]. Combining a confidentiality and an authentication mode together is error prone and likely to result in something that compromises the security properties of the encryption scheme, the authentication scheme, or both. Due to the known difficultly of creating AEAD modes, I looked for



(a) Encrypt-then-MAC (b) Encrypt-and-MAC (c) MAC-then-Encrypt

Figure 4-1: Approaches to AEAD modes (Reproduced from [2])

existing options as opposed to creating a new one.

In AEAD modes, the decryption operation combines integrity verification and source authentication into a single step. Typical approaches to AEAD modes are shown in Figure 4-1. In AEAD mode, implementations provides the following functions:

- Encryption
  - *Input*: message (the header and payload, where the header will not be encrypted but will be covered by integrity protection) and a key (potentially multiple keys depending on the AEAD implementation)
  - *Output*: ciphertext and a message authentication code (MAC)
- Decryption
  - *Input*: message header, ciphertext, tag, and key(s)
  - *Output*: plaintext, and indication that the tag does or does not match the supplied ciphertext and header

There are several AEAD modes of operation. I selected Synthetic Initialization Vector (SIV) mode [48], which has not been standardized by NIST, but has been standardized by the IETF in RFC 5297 [36].

SIV mode is a MAC-then-Encrypt (shown in Figure 4-1c) AEAD mode and works by first creating a MAC by applying CMAC [24] over the message header and plaintext payload. The MAC is then used as an initialization vector (IV) for encrypting the plaintext payload in Counter mode. Figure 4-2 describes and illustrates this process.

SIV mode is described in more detail in the specification by Phillip Rogaway [48].

One of the advantages of SIV mode is that it can be based on an arbitrary block cipher and the same underlying block cipher can be reused for encrypting and tagging. Another advantage of SIV mode is that it is deterministic and therefore does not require any randomness. Providing secure random values is a difficult requirement to satisfy for an RT-PSM system because computationally limited devices often do not have access to high-quality sources of randomness. It is possible to generate random numbers on an Arduino, for instance, by seeding a generator with input from an unconnected analog pin or by pre-computing a random blob with a high-quality PRNG and storing it in flash memory. These are not robust solutions. The first option offers a poor quality of randomness and the latter option would take up valuable program space.

A small disadvantage of SIV mode is that it requires two distinct keys for encryption and MAC. There are two newer modes that work with just one key: HBS (Hash Block Stealing) and BTM (Bivariate Tag Mixing) [37, 38]. The downside to these modes, apart from being newer and having undergone less peer review as a result, is that the modes each have their own MAC generation algorithms that cannot be built from the block cipher used for encryption. I decided not to select them to avoid increasing the code base and thus the flash memory requirements.

It is worth noting that SIV mode is resistant to timing-based and power-monitoring-based side-channel attacks [36]. During decryption the tag is verified after the ciphertext has been decrypted (shown in Figure 4-2), so the same amount of work is performed whether the tag is valid or invalid[9]. This is both a positive (in that it prevents side-channel attacks) and a negative (in that it increases the potential impact of denial of service attacks[10]). Also, the tag could be used to replace the CRC, but

[9]This requires that a secure implementation of `memcmp` be used for verifying the tag. The secure implementation must take a constant amount of time to compare two tags regardless for where the difference might be. If the `memcmp` routine takes a variable amount of time—for instance, by returning as soon as a difference in the tags is detected—it will be leaking information and could potentially be used as a vector for an attack.

[10]An attacker can take advantage of the fact that under SIV mode a device must first perform the entire decryption process before it discovers a message is invalid or not. This aspect of SIV mode can be used to amplify a denial of service attack. This risk can be mitigated by having a device turn

Table 4.3: Ciphers considered for a secure RT-PSM system, ordered from best to worst by the figure of merit (FOM)

Cipher	Year	Block Sizes[a]	Key Sizes	Patented	Figure of Merit
Speck [13]	2013	64, 96, 128	96, 128, 144, 192, 256	No	3.5
Simon [13]	2013	64, 96, 128	96, 128, 144, 192, 256	No	6.6
AES [22]	1998	128	128, 192, 256	No[b]	7.2
Robin [33]	2014	128	128	No	7.3
RC5 [46]	1994	64, 128	0-2040	Yes[d]	8.4
LBlock [60]	2011	64	80	No	9.1
TWINE [54]	2011	64	80, 128	No	13.5
IDEA [27]	1991	64	128	No[c]	-
TEA [59]	1994	64	128	No	-

[a] Only block sizes greater than 64 bytes are listed. Block sizes less than 64 bytes tended to require keys sizes with insufficient security (e.g., less than 80 bits).
[b] No, but specific implementations are patented (e.g., Analog Devices, Inc. owns a patent)
[c] Last patented by ASCOM TECH AG, expired in 2012
[d] Patented by RSA Security

because decryption must precede tag verification this might not be desirable. CRCs are short and can be verified as soon as a message is received.

Freshness. As mentioned above, freshness can be provided by including a unique nonce in each message header. A message receiving device will only process messages that include a nonce that differs from the nonces in past messages. Including a unique nonce in the message header provides necessary condition that messages with identical payloads will result in different ciphertexts[11], thus preventing adversaries from using pattern analysis and knowledge of the message protocol to discern message contents.

Block Cipher. AEAD mode requires a block cipher. Table 4.3 shows some of the block ciphers I considered. Stream ciphers cannot be used to provide integrity protection or authentication and were not considered.

off its radio after a certain number of invalid messages have been received in a certain time window.

[11]The nonce will alter the MAC which is used as the IV for Counter mode and will alter the entire ciphertext (shown in Figure 4-2).

All block ciphers considered were reputed to be lightweight in nature. With the exception of Simon, the ciphers in Table 4.3 were designed to perform well in software. The choice of block cipher was informed in a large part by [23] and [14] which evaluate the performance of the ciphers listed in Table 4.3 on AVR, MSP, and ARM hardware—three widely used types of low-power microcontrollers. Table 4.3 includes the Figure of Merit (FOM) metric for each cipher which was calculated in [23]. The FOM represents the average performance of a block cipher in terms of RAM usage, code size, and execution time on all of the AVR, MSP, and ARM microcontrollers.[12] A lower FOM score is better. In terms of code size, RAM usage, and execution time, [14] finds Speck to completely dominate. And [23] finds Speck best overall (as shown in the FOM score in Table 4.3), but slightly worse than AES in terms of execution time.[13]

Speck is a family of block ciphers recently introduced by the NSA and reported, as evidenced by the results in [14] and [23], to have excellent performance in software. While Speck has not been standardized, its development by the NSA provides it some legitimacy, which was an important factor in the choice of block cipher. The performance, security guarantees, and provenance of Speck make it a good choice for a RT-PSM system.

Key Provisioning. As mentioned above, keys will be provisioned before a mission. One method of doing this involves deriving all keys from a single seed. The benefits of this approach are that there is no need for a large key database and the implementation is quite simple. Given a pseudo-random function F (e.g., as described in [32]) which maps variable-length input to fixed-length output, keys can be provisioned as $F(\text{master key}, \text{unique identifier})$. In a tactical RT-PSM system this scheme would enable a combat medic to communicate with any soldier regardless of whether their devices were provisioned at the same time, provided that the medic device is allowed to know F and the master key. A disadvantage of this system is that it has a single

[12] Table 4.3 does not include a FOM for IDEA or TEA because [23] did not evaluate these ciphers.

[13] Although, the authors of [23] note that they compared an assembly implementation of AES to a C implementation of Speck.

point of failure. If the master key is lost so is all forward and backward secrecy; the damage extends to all missions (past and future). Key rotation can reduce this risk by limiting the amount of data tied to a single master key. There are many ways to do key rotation. I will suggest a simple one: at regular intervals, a new master key can be generated and distributed to all of the provisioning machines. Past keys can be deleted or stored securely.

To summarize, I found that an acceptable solution for providing the security requirements in Section 4.1 is to encrypt message payloads with Speck in SIV mode and to append the generated MAC over the encrypted payload and message header, which includes a nonce, to the end of messages.[14] Both key provisioning and key management in general are important and warrant more development in future work.

[14] A CRC is also included at the end of messages and covers the message header, the encrypted payload, and MAC.

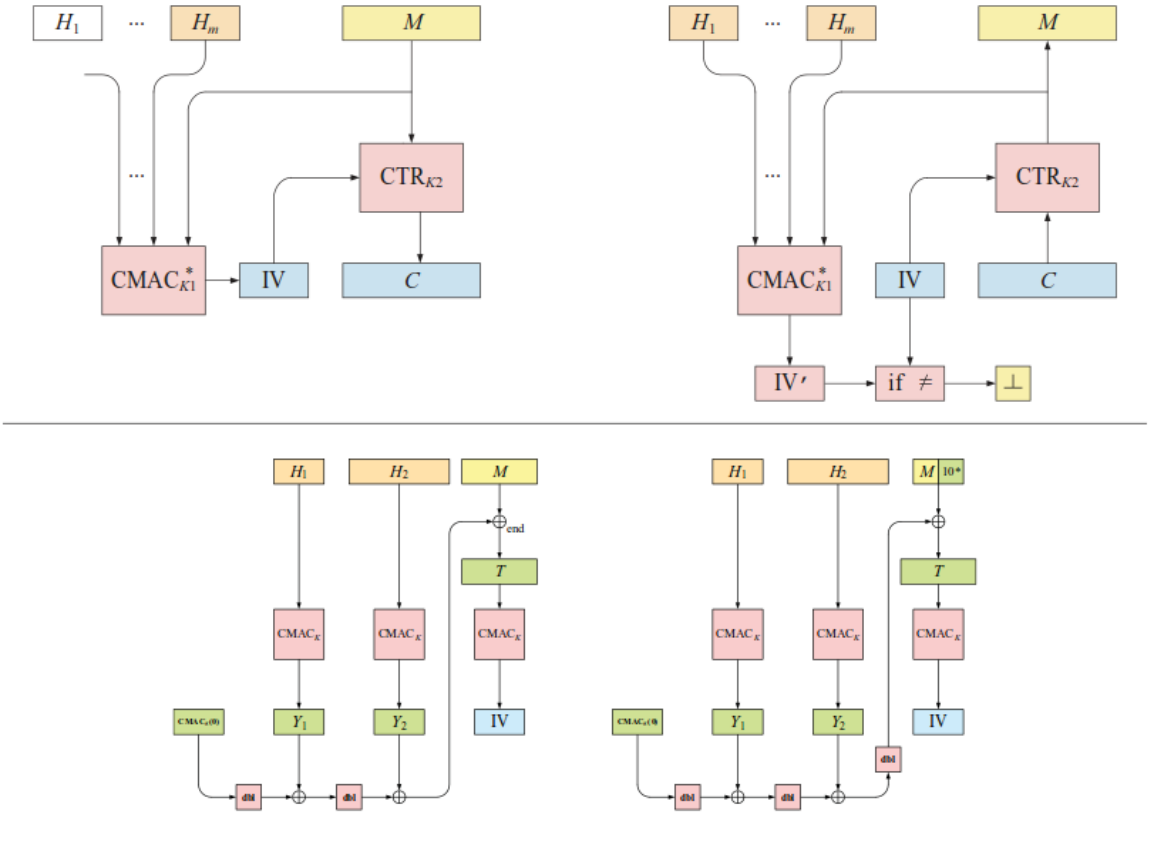
Figure 4-2: **Top**: Definition of SIV mode. **Middle**: Illustration of encryption (left) and decryption (right). **Bottom**: Illustration of CMAC when the final argument has n or more bits (left) and when it does not (right). (Diagram and caption reproduced from the short SIV specification [48])

Algorithm $\text{SIV-Encrypt}^{H_1,\ldots,H_t}_{K1\,K2}(M)$
if $t \geq n-1$ **then return** $\perp$
$\text{IV} \leftarrow \text{CMAC}^*_{K1}(H_1,\ldots,H_t,M)$
$C \leftarrow \text{CTR}_{K2}(IV, M)$
return $\text{IV} \parallel C$

Algorithm $\text{CMAC}^*_K(X_1,\ldots,X_m)$
$S \leftarrow \text{CMAC}_K(0^n)$
for $i \leftarrow 1$ **to** $m-1$ **do** $S \leftarrow \textbf{dbl}(S) \oplus \text{CMAC}_K(X_i)$
if $|X_m| \geq n$
then return $\text{CMAC}_K(S \oplus_{\text{end}} X_m)$
else return $\text{CMAC}_K(\textbf{dbl}(S) \oplus X_m 10^*)$

Algorithm $\text{SIV-Decrypt}^{H_1,\ldots,H_t}_{K1\,K2}(\mathbb{C})$
if $t \geq n-1$ **or** $|\mathbb{C}| < n$ **then return** $\perp$
$\text{IV} \leftarrow \mathbb{C}[1..n],\ C \leftarrow [n+1..|\mathbb{C}|]$
$M \leftarrow \text{CTR}_{K2}(\text{IV}, C)$
$\text{IV}' \leftarrow \text{CMAC}^*_{K1}(H_1,\ldots,H_t,M)$
if $\text{IV} = \text{IV}'$ **then return** M **else return** $\perp$

Algorithm $\text{CTR}_K(\text{IV}, M)$
$Ctr \leftarrow \text{IV}\ \&\ 1^{n-64}\ 01^{31}\ 01^{31}$
$Pad \leftarrow E_K(Ctr)\ E_K(Ctr{+}1)\ E_K(Ctr{+}2)\ \cdots$
return $M \oplus Pad\,[1..|M|]$



THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

AUTOBAN Proof-of-concept Implementation

In this chapter, I discuss the AUTOBAN proof-of-concept implementation. The AUTOBAN proof-of-concept implementation is intended to demonstrate how to protect RT-PSM systems from the threats detailed in Section 3.3 using the solution in Section 4.3. Due to time constraints, the AUTOBAN proof-of-concept implementation focuses on showing successful defenses against two of the main categories of the top threats identified in Table 3.4: active attacks (*A1-A4*) (e.g., identity spoofing) and passive attacks (*P3,4*) (e.g., opportunistic privacy compromise).

Before describing the proof-of-concept implementation and attack mitigations in more detail, I will first briefly describe differences between the proof-of-concept implementation and a standard RT-PSM system (see Chapter 2).

5.1 Implementation Components

In order to reduce development effort, the AUTOBAN proof-of-concept implementation differs from the RT-PSM system described in Chapter 2. The proof-of-concept implementation is intended to demonstrate the applicability of the security mechanisms derived in Chapter 4.3 to the specific RT-PSM use cases defined in Chapter 2. The eventual deployment scenario would more closely mirror the OBAN RT-PSM

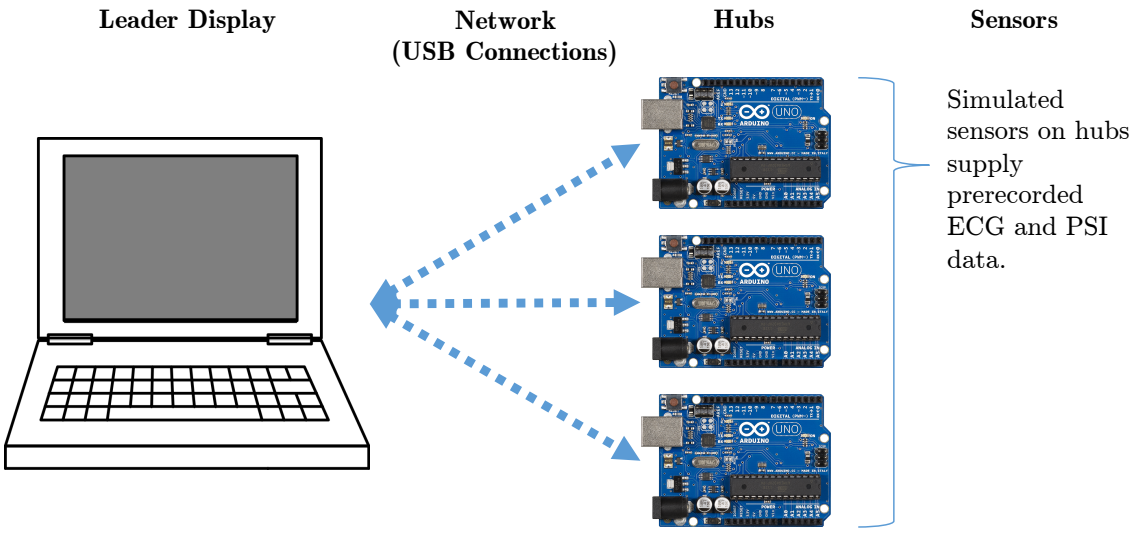


Figure 5-1: AUTOBAN proof-of-concept implementation overall system diagram

system. The AUTOBAN proof-of-concept implementation consists of the same core components identified for RT-PSM systems in Section 2.1 (e.g., sensors, hubs, and displays (or EUDs)), but differs in two important ways. First, radio-frequency based communications are not used; the devices communicate over Ethernet or serial connections. Using Ethernet or serial as the underlying network layer preserves the broadcast nature of the RF-based communication medium of a standard RT-PSM system. That is, all parties with a *Receive RF* capability can see all transmissions. Second, simulated sensors are used in place of actual sensors. Simulated sensors on the hub device include Physiological Strain Index (PSI) and ECG monitors. The sensors are mocked out and use pre-recorded data that is stored in the flash memory of the devices. Figure 5-1 provides an overall system diagram of the proof-of-concept implementation.

The hub devices are implemented using Arduino Uno boards based on an ATmega328P microcontroller (described in Table 5.1). The Arduino Uno was selected for two main reasons: first, the hardware is similar to that found in actual RT-PSM systems, and second, the ATmega328P on the Uno comes with a bootloader, providing the ability to upload new code by USB without the need for an external hardware programmer.

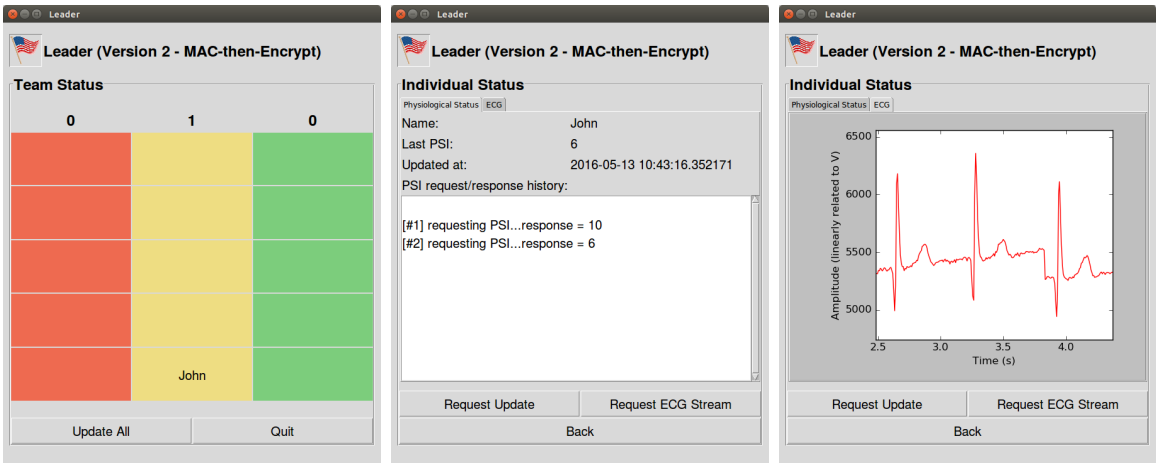
The Arduino Uno device that is used in this implementation is more resource

	AUTOBAN proof-of-concept hub	**OBAN reference implementation hub**
Model	ATmega328P	ATmega2560
CPU	8-bit RISC	8-bit RISC
Frequency (MHz)	16	16
Registers	32	32
Architecture	Harvard	Harvard
Flash (KB)	32	256
SRAM (KB)	2	8
EEPROM (KB)	1	4

Table 5.1: Key characteristics of the hub microcontroller for the AUTOBAN proof-of-concept implementation and the OBAN reference implementation

constrained than the Arduino-based Seeeduino board (described in Table 5.1) used in the OBAN reference implementation [19]. A more resource-constrained board was chosen to highlight the fact that the lightweight cryptographic primitives used in the cryptographic library can work on very limited hardware.

A computer is used to simulate the leader's EUD device. Figure 5-2 shows the different screens in the leader display application. The interface is written in Python and is meant to simulate the same features as the leader EUD in the OBAN demo application (shown in Figure 2-2). The first screen (Figure 5-2a) provides a summary of the PSI levels of the team members. The different colors indicate a squad member's risk for heat stroke. On the bottom of the home screen there's a button that gives the leader the ability to request a PSI reading from all group members. Clicking on a box with a group member's name opens up the individual summary screen (Figure 5-2b) which provides more details on the selected squad member's PSI. PSI values range between 1 (low) and 10 (high). The most recent PSI value is shown at the top of the screen under the squad member's name and the past history of responses is shown in the middle of the screen. Finally, clicking on ECG tab in the individual screen opens a new tab within the screen with a plot showing any collected ECG data (Figure 5-2c). At the bottom of the individual screen there are buttons allowing the leader to request a PSI update (pull mode) or to request an ECG stream (stream



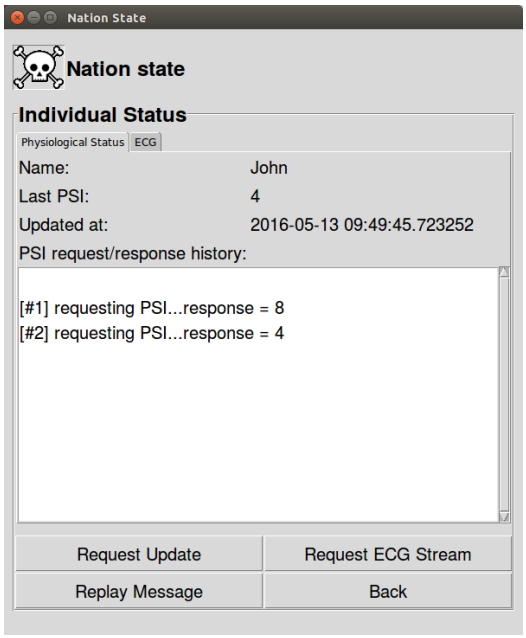
(a) Group screen (b) Individual PSI screen (c) Individual ECG screen

Figure 5-2: Screen shots of the leader display application

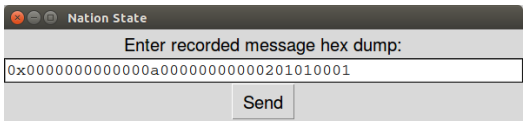
mode).

The AUTOBAN proof-of-concept implementation also includes simulated interfaces for nation state and hacktivist attackers. The interfaces are used for demonstrating attacks on the system. Like the leader display, the nation state and hacktivist interfaces are also written in Python and run on a computer. Figure 5-3 shows the screen shots of the two applications. The nation state interface (5-3a) is nearly identical to the leader display interface. This emphasizes that nation states have lot of resources and would have no difficulties procuring *Send Message* and *Receive Message* capabilities. The main difference between the leader display and the nation state interface is that the nation state interface has no underlying keying material and cannot decipher or create valid encrypted messages[1]. The nation state can, however, replay encrypted messages that it has recorded, as will be seen later. The hacktivist is a less sophisticated entity than the nation state and consequently has a less sophisticated interface (Figure 5-3c) that only allows it to see all message traffic in the network.

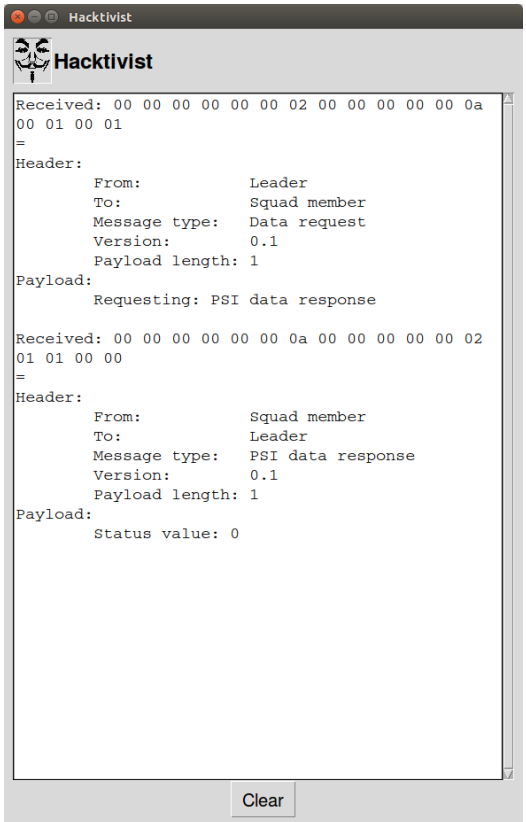
[1]It can, of course, send a garbled message (e.g., *A6* denial of service attack).



(a) Nation state individual screen



(b) Nation state replay interface



(c) Hacktivist message traffic screen

Figure 5-3: Screen shots of the nation state and hacker display applications

5.2 Cryptographic Library

As discussed in Section 4.3.6, in order to prevent the top attacks in Table 3.4 the best option is to use a block cipher in an Authenticated Encryption with Associated Data (AEAD) mode of operation. The most promising block cipher and AEAD mode based on the resource constraints of the hardware components is Speck in SIV mode (Speck-SIV). I implemented Speck-SIV for the Arduino hub devices and the leader display application.

The AVR chip on the Arduino makes use of the `avr-gcc` complier and the AVR Libc library, allowing development in C and C++. I implemented Speck-SIV mode in C++. It is worth noting for others developing applications for runtime-sensitive environments, like the ATmega328P AVR chip, that care should taken to avoid unwanted side effects of the C++ calling conventions. Copy constructors, for instance, which may occur on function invocation, can take up considerable time and memory. It may be necessary to remove the possibility for these calling conventions (e.g., using the C++11 `delete` keyword) or to inspect the generated assembly code.

I implemented multiple versions of the underlying Speck cipher. They are discussed in more detail in Chapter 6.2.1. The performance characteristics are also given in Chapter 6.2.2. The version included in the proof-of-concept implementation was designed to support the push, pull, and stream modes described in Chapter 2.2.

The Speck-SIV implementation operates on 128-bit blocks and keys. A 128-bit key was chosen because it provides an adequate amount of security without needing excessive amounts of storage. The 128-bit block size was chosen in order to compare the performance of Speck with the performance of AES (Chapter 6.2.2). With a 128-bit key, Speck supports a 64-bit or 128-bit block size. AES only supports 128-bit blocks. In general, I would recommend selecting a block size that is close to the minimum message payload size in order to avoid extra computation on the hubs. Larger blocks typically require more compute cycles to manipulate. This is certainly true for the Arduino's 8-bit processor and is evidenced in the Speck benchmarks on the platform [58]. The keys were pre-generated and hardcoded into the Speck-SIV

library in order to simulate key pre-placement for each demonstration scenario. In a real deployment, keys would be generated during the device provisioning stage before a mission and a configuration protocol would be required to load the keys onto the devices.

5.3 End-to-end AUTOBAN Demonstration

As mentioned above, the proof-of-concept implementation is intended to show how an RT-PSM system can reduce the likelihood of a subset of the top threats: passive attacks (*P3,4*) and active attacks (*A1-A4*) . In this section, I describe specifically how the AUTOBAN proof-of-concept implementation can successfully defend against opportunistic privacy compromise (*P4*) and on-demand privacy compromise (*A3*). I will first motivate the defense techniques by describing successful attacks on a version of the system that is similar to the OBAN reference implementation in that it uses a message protocol with no security mechanisms. After the attacks are described, versions of the system with message protocols capable of defending against the attacks are presented. Overall, there are 3 versions of the AUTOBAN proof-of-concept implementation's messaging protocol:

1. **Plain Messages** — No security mechanisms (top of Figure 5-4)
2. **Encrypted Messages** — Payloads are encrypted (middle of Figure 5-4)
3. **MAC-then-Encrypt Messages** — Message payloads are encrypted and messages include a MAC over the message header and payload. (bottom of Figure 5-4)

In the Plain Message version of the protocol functioning under the pull-style use case, the leader device requests data from a squad member in the clear and the squad member replies in the clear. Figure 5-5 illustrates this process. This is the most common use case and I will refer back to it exclusively throughout this section while describing attacks. The attacks would work in a similar manner with respect to the push and pull style use cases, but for brevity I only describe them with respect to the pull-style use case.

Figure 5-4: AUTOBAN proof-of-concept message protocols diagram shown with an example 16-byte payload

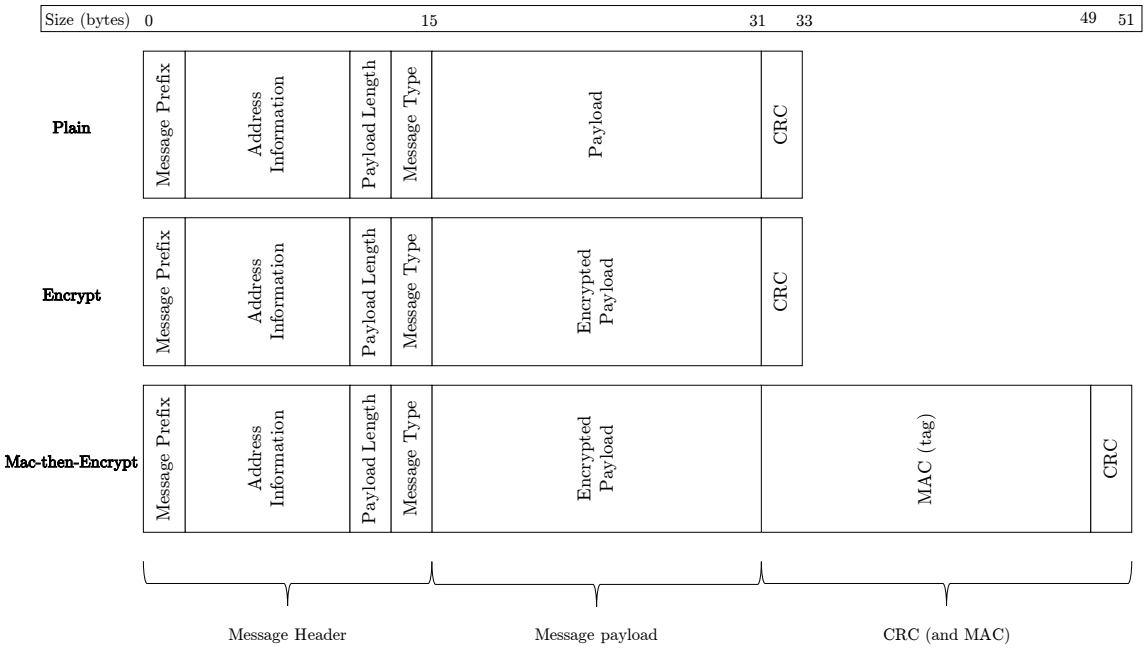


Figure 5-5: Plain Message version of system in the pull-style use case

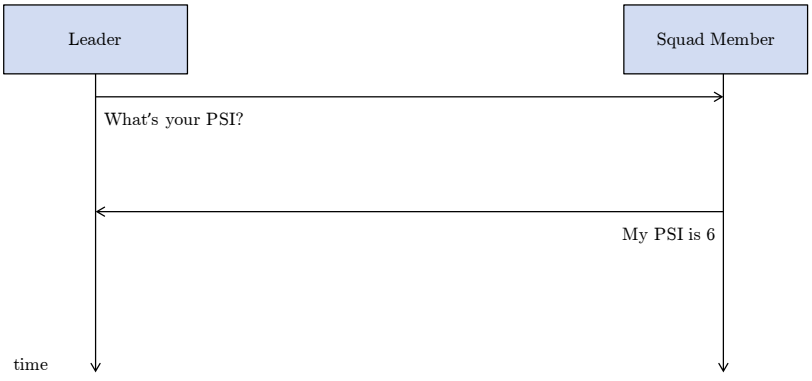


Figure 5-6: Privacy compromise ($P4$) — Plain Message version vs. hacktivist

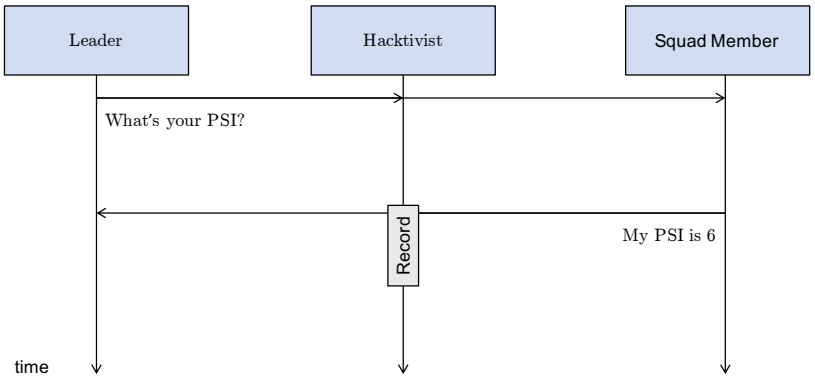
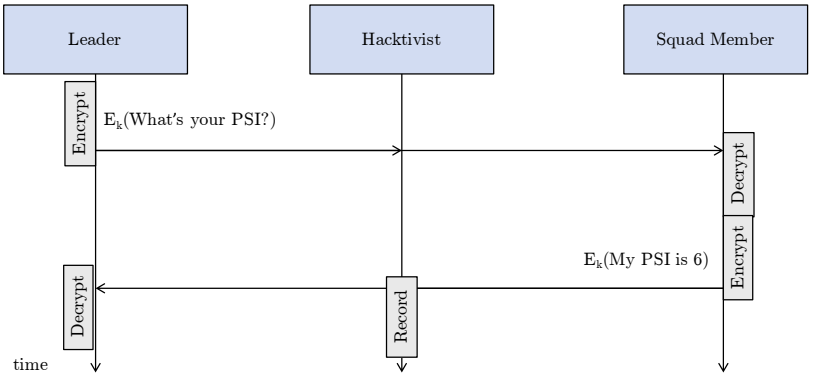


Figure 5-7: Defense against privacy compromise ($P4$) — Encrypted Message version vs. hacktivist

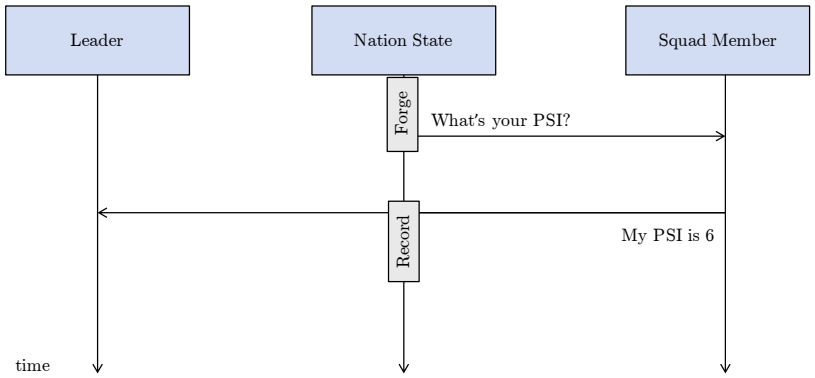


Opportunistic Privacy Compromise

In the opportunistic privacy compromise attack, a hacktivist attacker is able to exploit the broadcast nature of the communication medium and the unencrypted quality of the message traffic in the Plain Message protocol. While hacktivists are not seen as particularly sophisticated attackers, they are able to acquire the *Receive Message* capability. In this case, the hacktivist is able to see the messages sent between the leader and hub devices, as shown in Figure 5-6.

The Encrypted Message version of the protocol can defend against the hacktivist attack by encrypting the message traffic between leader and squad member devices, as shown in Figure 5-7. The leader and squad members are unable, as in Plain Message version, to detect the presence of the eavesdropping hacktivist, but no longer need to be concerned about the content of their messages being read. The hacktivist attacker

Figure 5-8: Identity spoofing (*A3*) — Plain Message version vs. nation state



will be unable to decipher the contents of the messages without the cryptographic keying material.

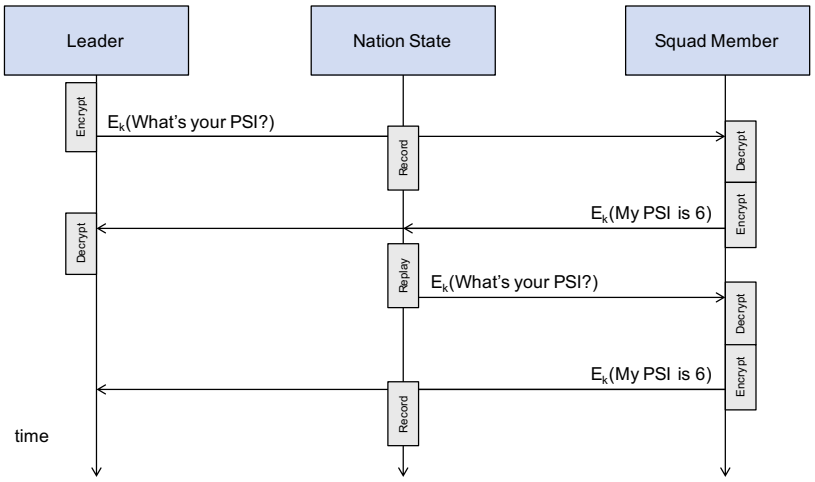
There are two main drawbacks to the Encrypted Message version of the protocol. First, the header information is still sent in the clear, so as to allow squad members and the leader to quickly disregard messages not intended for them. This information could be used by the hacktivist to identity devices in the RT-PSM network (*P2. device identification*). Second, if the message contents are identical, the ciphertext will also be identical, allowing the hacktivist an opportunity to use pattern analysis techniques to potentially discern message contents.

The first issue highlights a trade-off between sustainability and security: hubs and leader devices can quickly stop processing messages in legitimate cases to preserve battery life at the cost allowing adversaries to see header information. The second issue will be resolved in the MAC-then-Encrypt Message version of the messaging protocol.

Identity Spoofing

In an identity spoofing attack (*A3*), a nation state attacker spoofs the identity of a message sender. In Plain Message version of the protocol, a nation state attacker could simply craft a message, as shown in Figure 5-8, providing them an avenue for privacy compromise at will (*A3*). In the Encrypted Message version of the protocol, mounting a identity spoofing attack is not as easy. A nation state cannot create a

Figure 5-9: Identity spoofing ($A3$) — Encrypted Message version vs. nation state

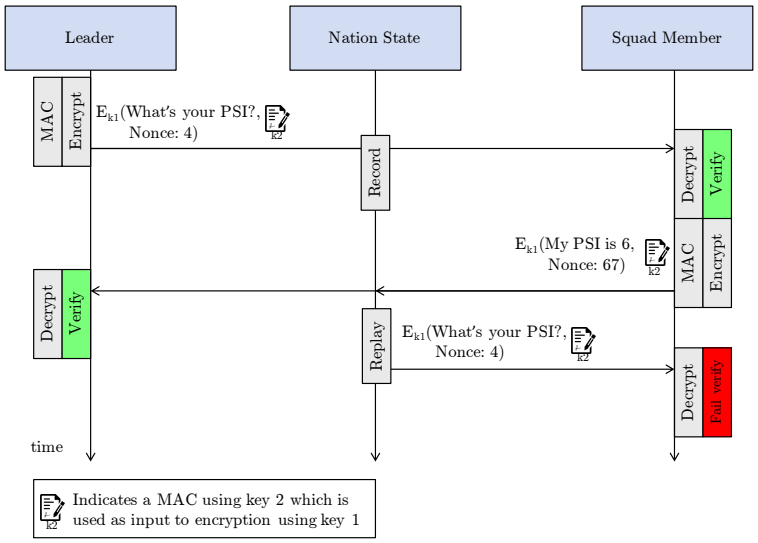


valid encrypted message and must perform a replay attack—first recording a valid message, and later replaying it, as shown in Figure 5-9. In this case, the nation state attacker is unable to decipher the received messages, but being able to request ciphertext at will enables pattern analysis as well as a battery drain attack ($A5$).

The MAC-then-Encrypt Message version of the protocol can protect against both versions of the identity spoofing attack by providing source authentication and message integrity. When sending a message, a leader or hub first generates a tag based on the message header and the message payload. The tag is then used as an initialization vector for encrypting the message payload in Counter mode (this is SIV mode, see Figure 4-2). A unique nonce is included in the message header to provide a freshness property, ensuring identical payloads map to unique ciphertexts each time. The message recipient will disregard any messages with repeat nonces. The encryption prevents attackers without the proper cryptographic keying material from generating anything other than garbled messages, and the nonce and MAC prevent replay attacks. Figure 5-10 shows this process in action against the same identity spoofing attack the was successful in Figure 5-9. Keeping track of past nonce values can be implemented efficiently by using random values (and a bloom filter), timestamps, or simply a counter. The AUTOBAN proof-of-concept implementation uses a counter.

This section has shown how the security mechanisms derived in Chapter 4.3.6 can defend against a subset of the top active and passive threats to RT-PSM systems.

Figure 5-10: Defense against identity spoofing (*A3*) — MAC-then-Encrypt version vs. nation state



A few open issues, such as how to protect against certain attacks (e.g., *P2. device identification*) and how to provision devices with keys, are left to future work.

Chapter 6

Performance Evaluation

In this chapter, I analyze the performance of my implementation of SIV mode in combination with a few underlying ciphers. To keep the measurements repeatable and meaningful, I adhere to a common framework detailed in Section 6.1. Section 6.2 presents metrics relevant to evaluating the performance of block ciphers in ECB and SIV modes on low-power devices.

6.1 Evaluation Methodology

In order to keep this performance analysis comparable to past work and to ensure repeatability, I adhere to a common framework detailed in [14]. The framework provides two important pieces of information. First, it describes at a high level what task the cipher is expected to perform. Second, it provides performance metrics for comparison. The performance metrics allow various ciphers to be ranked against one another with respect to the same application and hardware platform.

6.1.1 Application Types

When making comparisons between lightweight cryptographic primitives it is important to understand the application needs. Following [14], there are four main types of applications relevant to lightweight cryptography in sensor networks. The appli-

cations differ along two dimensions: the lifetime of the cryptographic keys and the amount of data being exchanged.

With respect to the lifetime of the key(s) the framework makes the following distinction:

- *Fixed Key* — A fixed key application makes the assumption that the key or keys used with the encryption scheme rarely, if ever, change. In software, the code required to expand the key into the key schedule can be omitted. Additionally, the key does not need to be stored; instead the expanded key schedule can be precomputed and stored in flash memory. A fixed-key assumption is applicable to tactical RT-PSM systems that have no requirement to re-key during a 7-day mission, or lack the design flexibility or program space to allow for a re-keying scheme.
- *Flexible Key* — With a flexible key, it is assumed that the keying material is changed often. Under this assumption, a tactical RT-PSM system would be expected to change keys multiple times over the course of a mission. This would be necessary if the system needs to be protected from device capture (full-scope attacks *F2-6* from Chapter 3) or if the data exchanged by system components is valuable enough to warrant the development effort and power expenditure required for a key change.

With respect to the amount of data being transmitted, the following distinction is made:

- *Small Data* — For small data size comparisons, it is assumed that a single block of data (i.e., 128-bits in the AUTOBAN proof-of-concept implementation) is encrypted at a time. The amount of data encrypted could be less than one block, but it's not useful to draw a distinction there as it would incur the same cost as encrypting a full block. The total stream of data is also small—small enough to make any setup costs relevant.
- *Large Data* — For large data comparisons, it is assumed that the length of the data stream is long enough to effectively amortize away any setup costs (e.g. key schedule expansion).

The four types of application are *Fixed Key/Small Data*, *Fixed Key/Large Data*, *Flexible Key/Small Data*, and *Flexible Key/Large Data*. For the target use case presented in Chapter 2, Fixed Key/Small Data and Fixed Key/Large Data are the most appropriate models. Some quick notes about these two combinations: Fixed Key/Small Data applies to low data rate applications that require authentication (e.g., the pull-style updates and notifications use cases described in Chapter 2.2), and Fixed Key/Large Data is applicable to certain streaming-based sensor applications (e.g., the streaming use case described in Chapter 2.2). The analysis will focus on these two types of applications. Analysis of cryptographic primitives serving the other two types of applications, Flexible Key/Small Data and Flexible Key/Large Data, would be similar, but more concern would be given to setup costs and re-keying methods.

6.1.2 Performance Metrics

I collected four types of performance metrics: code size (flash), memory usage (RAM), execution time (cycle counts/cost), and energy consumption. I considered the first three metrics, flash, RAM, and cycle counts, because they provide insight into the block cipher's characteristics with respect to computationally-limited devices and cannot be inferred through other measurements. The same is not true of energy consumption. The energy consumption metric is redundant in the sense that it can be computed given cycle counts and the device's energy model. I include it since it is often preferred over cycle counts because devices in tactical RT-PSM systems (and more generally BANs) have tight energy budgets. From the first three measures, I derived an overall performance metric, *rank*. Below I describe how each metric was extracted and how to compute *rank*.

Code size (flash)

Code size is a measure of the amount of information that is stored in the flash memory of the device. I refer to this metric as *flash*. Flash memory is a limited resource on

embedded devices. For instance, as mentioned in Table 5.1, the ATmega328P microcontroller used in the AUTOBAN proof-of-concept implementation only has 32KB of flash memory. It would be reasonable for an application to use microcontrollers with even less flash memory.

To calculate the code size of a binary built for the ATmega328P microcontroller, I used the GNU `avr-size` tool (version 2.24). The `avr-size` tool takes a binary file and lists the section sizes in bytes. The binary code size reported in the results is the sum of a binary file's `text` and `data` sections. The `text` section of the binary corresponds to all the machine instructions the ATmega328P microcontroller is going to execute. The `avr-size` tool considers the `prgmem` section—a section of the binary which includes constant data that will be stored in flash and referenced from flash during program execution—to be part of the `text` section[1]. The `data` section of the binary contains all of the global initialized variables declared in the program. The implementations in this analysis do not use global uninitialized variables, so the `bss` section size is 0 and is ignored.

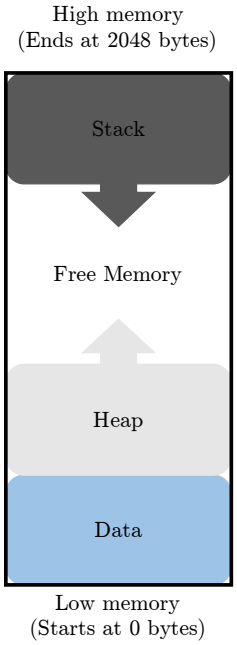
Memory usage (RAM)

RAM consumption is composed of two parts: stack and data consumption. The stack consumption corresponds to the information involved in the active subroutines of the program's execution and naturally varies during a program's execution. At the start of execution, the `data` section of the binary is loaded from flash into RAM—this gives the data consumption portion of memory usage (and it can be calculated with the `avr-size` tool as discussed above). Together, stack and data usage might include the information to encrypt, keys, round keys, initialization vector, etc. Figure 6-1 shows the memory operation and layout of an Arduino Uno. Heap is included in the diagram, however, none of the cipher implementations considered in this analysis involve dynamic memory allocation, so no heap consumption is reported.

RAM usage is generally the limiting factor for applications on embedded devices.

[1]So, for instance, if an AES implementation has S-boxes stored in program memory (`prgmem`), they are included in the flash size, as they should be.

Figure 6-1: Arduino Uno RAM operation



Understanding the RAM usage of a cryptographic primitive is vital to successfully integrating it into an application.

I experimented with measuring stack usage in two ways: 1) (on board) measuring the number of bytes between the bottom of the stack and the top of the heap and 2) measuring the stack height using the Avrora simulator [56] and stack monitoring tool.

Method 1 made use of the following code snippet provided by [25]

```
// Measure distance between stack and heap
int freeRam ()
{
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start :
      (int) __brkval);
}
```

which reports the difference in bytes between the bottom of the stack and the top of the heap (i.e., the *Free Memory* region in Figure 6-1). By sprinkling calls to `Serial.println(freeRAM());` throughout the code I was able to get an idea of the

RAM usage patterns throughout program execution. Knowing where to place the calls required familiarity with the source code and the underlying encryption and decryption algorithms. My recommendation for future developers trying to apply a similar method would be to inspect a call-graph of the program's execution in order to understand where the stack usage will be the highest and thus where to place calls to the `freeRam()` method.

For method 2, I measured stack consumption using the Avrora tool (version 1.7.117). Avrora is a simulation and analysis framework for AVR microcontrollers created as a research project by the UCLA Compilers Group [56]. With the Avrora tool, I can inspect the height of the stack during the program's execution.

In a few preliminary trials, I was able to get very similar results using these different methods. The results reported in the following section were taken using the latter method. I chose the Avrora tool because it didn't involve modifying the code and produced results faster than method 1.

Runtime (cycle counts/cost)

Cycle count indicates the number of CPU clock cycles spent on executing a certain routine. In this analysis, *cost*, which is the number of cycles it takes to transform one byte of plaintext into one byte of ciphertext (or vice versa), is of specific interest. The cycle count required to encrypt a byte of data determines what types of applications can be supported by the cryptographic primitive. For instance, if the number of cycles required to encrypt a byte of data is too large, real-time, high-throughput streaming applications cannot be supported.

As with RAM usage, I considered measuring the cycle counts in two ways: 1) (on board) estimating the ticks using the execution time of the encryption (or decryption) routine and 2) using the Avrora simulator.

To find the cycle count using the first method, I first calculated execution time using the `millis` function[2] from the Arduino library. The function returns the number

[2]The Arduino library also includes a `micros` function that has a resolution of 4 microseconds. I did not use the `micros` function because the measurements did not require this level of precision.

of milliseconds since the Arduino board began executing. The execution time is computed as the absolute difference between the time returned by `millis` at the end of the encryption or decryption operation and at the beginning. Execution time is then multiplied by the clock speed to get an approximate cycle count.

The Avrora simulator can simply be set to display the number of cycles between two instructions.

In a few trials, both methods reported nearly identical results. I present results taken using the Avrora simulator method as it was the quicker of the two and did not involve instrumenting the code.

Energy

Energy measures the number of joules consumed by the device during the execution of a particular routine. In this analysis, I refer to energy as the number of micro-joules per byte (μJ/byte) required to transform plaintext into ciphertext (or vice versa in the case of decryption). Understanding energy consumption will allow developers to predict the battery lifetime of their devices. For a tactical RT-PSM system that must support data collection and transmission for a period of up to 7 days, understanding energy consumption is of vital importance. I considered two options for measuring power consumption.

The first option was to use a power-measurement board. In [26] the authors place a 22-Ohm shunt resistor between the Vdd pin and the 5-volt power supply, set triggers to indicate the start and end of the encryption routine. In that setup, the authors average the results from the measurement board over many runs of the encryption routine on randomly generated plaintext samples.

The second option is considerably easier, once again involving the Avrora tool. The Avrora tool includes an energy analysis monitor that uses Olaf Landsiedel's work on AEON to accurately estimate the power consumption of the executing AVR code [40]. I chose this option due to limited time and AEON's reported success.

Rank

I adopted a generalized version of the performance metric, *rank*, used in [14]. The metric is applicable to cryptographic routines implemented in software. A higher *rank* value is better. The measure is proportional to throughput divided by memory usage, and is given by:

$$rank = \frac{c/(w_c \cdot \text{cost})}{1 + w_f \cdot \text{flash} + w_r \cdot \text{RAM}}$$

where the weights, $W = (w_c, w_f, w_r)$, are the penalties given to cost, flash usage, and RAM usage respectively, and c is a constant equal to 10^8. The constant c has units of cycles and is used to ensure the overall *rank* metric is unitless and has a reasonable magnitude. In "Simon and Speck Block Ciphers on AVR 8-bit Microcontrollers". Beaulieu et al. set $w_c = 1$, $w_f = 1$ and $w_r = 2$ [14]. Setting $w_r > w_f$ symbolizes that on the device in question RAM is a more precious resource than flash. In this analysis, I offer rank values for the following sets of weighs: $W_{\text{cost}} = (2, 1, 1)$ and $W_{\text{RAM}} = (1, 1, 2)$[3]. Depending on the priorities of the application one mix of parameters may yield a more relevant *overall* metric of performance than others. Energy is not included in this metric because it can be inferred from cost. If power consumption is the top concern, for instance, a proper setting of the parameters could be $W = (2, 1, 1)$.

6.2 Cipher Evaluation

In this section, I describe the results of testing 5 ciphers according the methodology described in Section 6.1 in ECB and SIV mode for the ATmega328P. In the case of cost and energy, each of the reported results were obtained by averaging the outcomes of 100 measurement runs with random plaintext (or ciphertext) as input.

I did not implement all of the discussed ciphers from scratch since that was outside

[3] $W_{\text{flash}} = (1, 2, 1)$ was not considered because on the ATmega328P the implementations were in no way flash limited.

the scope of this research. In a few cases, I modified available existing implementations so that they worked with my implementation of SIV mode and built for the ATmega328P. These ciphers have been studied on this platform in previous work, but not under SIV mode or specifically in the context of an RT-PSM system [23, 14, 26].

There are many existing implementation of these ciphers. Some of them may have better performance along one or more of the dimensions considered here. I believe, however, that this framework should provide insight into how to think about choosing lightweight cryptographic primitives for a RT-PSM system. If better implementations are developed, they could be measured under this framework to better understand the absolute performance values for these ciphers under SIV or other AEAD modes of operation.

First I introduce the block ciphers I measured. Then, I present and discuss the results of the benchmarks.

6.2.1 Ciphers Measured

This analysis evaluates the following ciphers in ECB and SIV modes:

Original Speck implementation (`SPECK-OG`)

This is my original implementation of the Speck cipher. The implementation was developed with no optimizations to ensure correctness and built as a learning exercise to familiarize myself with the Speck cipher. I include it to provide insight into how important the right implementation is for an application's performance. This implementation stores the key schedule in RAM. `SPECK-OG` is not included in the performance graphs because it is not competitive with the other ciphers.

Optimized Speck implementation (`SPECK-OPT`)

This is my optimized implementation of the Speck cipher (the optimizations are described in Chapter 7). The implementation is a modified version of my original Speck cipher that was changed to support the AUTOBAN reference implementation's streaming mode of operation. Like `SPECK-OG`, this implementation stores the key schedule in RAM.

Optimized Speck implementation using EEPROM (`SPECK-EEPROM`)

This implementation is identical to `SPECK-OPT` except that the key schedule is expanded into EEPROM. EEPROM is flash memory on the ATmega328P that allows byte-level reading and writing by the executing program. The EEPROM memory is preserved when the board is turned off.

ASM Speck implementation (`SPECK-ASM`)

This is an assembly implementation of Speck taken from [58] that has been optimized for the Arduino platform. It unrolls calls to the round function and inlines function calls whenever possible. The key schedule is stored in RAM.

AES implementation (`AES`)

This is an implementation of AES also taken from [58] that has been optimized for the Arduino platform. The static S-box tables and other similar constant global variables have been placed into program memory in order to reduce RAM usage.

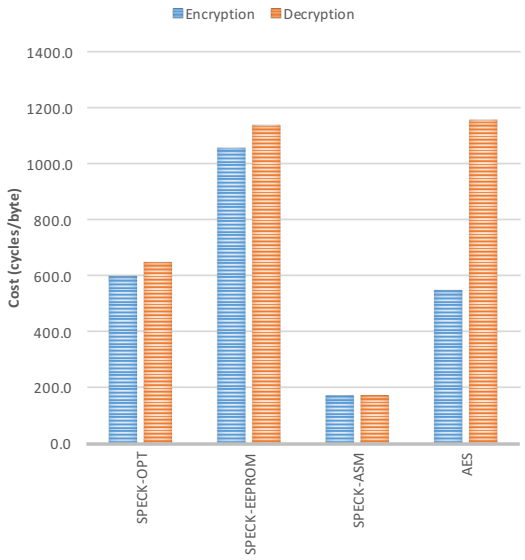
For all ciphers, the code was written following the 128-bit key/128-bit block size specifications for the cipher. AES was selected for two main reasons. First, it is approved by the NSA and is the NIST and ISO/IEC standard cipher. Second, in past benchmarks it was the closest competitor to Speck [23, 14].

6.2.2 Benchmarks and Discussion

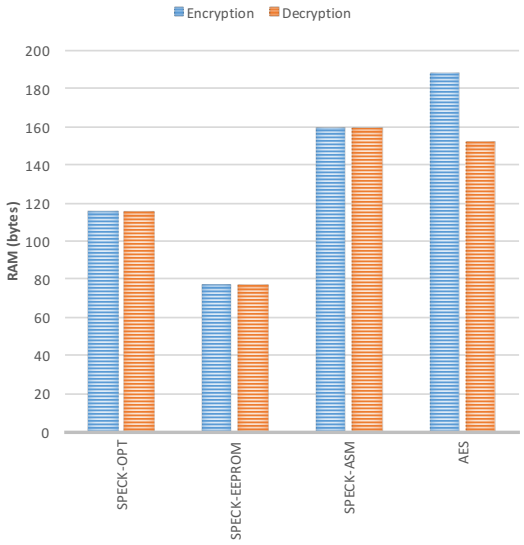
Table 6.1 and Table 6.2 show the metrics for each cipher in ECB and SIV mode, respectively. Both tables contain the results for the ciphers' performance over 16 bytes of text input (and 16 bytes of header input in the case of SIV). The measures in the SIV table also account for the MAC creation and verification routines. Figures 6-2 and 6-3 show plots of the cost and RAM metrics. `SPECK-OG` is excluded from the plots because it is not competitive with the other ciphers.

Among all of the block ciphers under my SIV implementation, `SPECK-ASM` ranks in the top spot for both the cost-prioritized ($W = (2, 1, 1)$) and the RAM-prioritized ($W = (1, 1, 2)$) settings of the weights. This was expected given the performance of SIV mode is entirely dependent on the performance of the underlying block cipher

Figure 6-2: Charts of cipher performance in ECB mode (16 bytes)

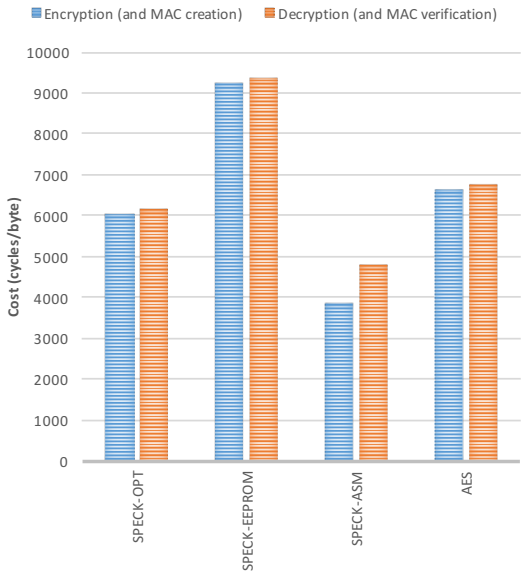


(a) Encryption/decryption costs

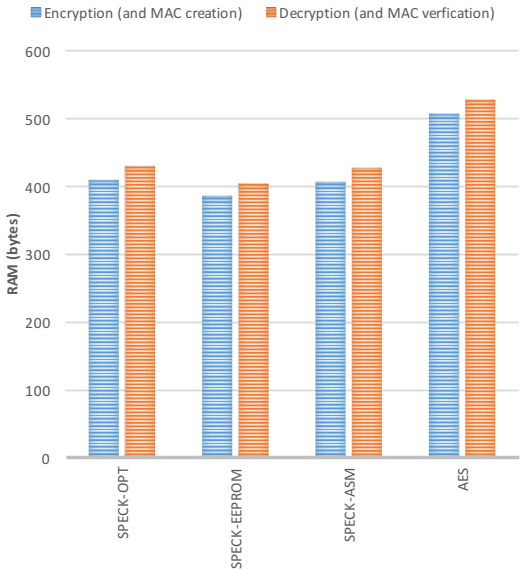


(b) RAM usage of ciphers

Figure 6-3: Charts of cipher performance in SIV mode (16-byte message payload and 16-byte message header)



(a) Encryption/decryption costs



(b) RAM usage of ciphers

Results using ECB mode (on 16 bytes)						
Cipher	**Flash** (Bytes)	**RAM** (Bytes)	**Cost** (cycles/byte)	**Energy** (μJ/byte)	**Rank 1** (W_{cost})	**Rank 2** (W_{RAM})
I: Encryption (no key schedule generation)						
SPECK-ASM	998	159	**169**	**0.5**	**448.8**	**255.2**
SPECK-OPT	**492**	116	598	1.7	230.4	137.2
AES	2947	188	547	2.6	55.0	29.1
SPECK-EEPROM	2186	**77**	1055	3.2	40.5	20.9
SPECK-OG	1226	153	75777	214.8	0.9	0.5
II: Decryption (no key schedule generation)						
SPECK-ASM	992	159	**175**	**0.5**	**435.9**	**248.0**
SPECK-OPT	**432**	116	647	1.8	232.5	140.8
SPECK-EEPROM	2246	**77**	1133	3.5	36.8	19.0
AES	2863	152	1154	3.1	27.4	14.4
SPECK-OG	1258	267	161316	457.7	0.3	0.2
III: Encryption and Decryption (including key schedule generation)						
SPECK-ASM	**1296**	463	**840**	**2.4**	**53.6**	**33.8**
SPECK-OPT	1372	404	2099	6.0	21.8	13.4
SPECK-EEPROM	2650	**156**	3729	10.9	9.1	4.8
AES	3417	396	4517	14.7	5.3	2.9
SPECK-OG	2990	637	309742	878.9	0.08	0.0

Table 6.1: Cipher performance evaluation using ECB mode

and other benchmarking efforts (as mentioned in Chapter 4.3.6) found Speck to be superior to AES in terms of flash, RAM, and cost [14, 23].

The results show that the top-ranked ciphers, SPECK-ASM, SPECK-OPT, and AES, are very similar in terms of RAM usage, which makes flash and cost the primary factors that determine the overall rank. As a result, the rank of AES is much lower, due to its large flash size. A large flash size is not unique to the particular implementation of AES I picked to benchmark. As confirmed by [23], AES implementations are larger than Speck implementations, mostly due to the space required for the S-boxes[4] and the lookup tables for round constants.

[4]S-box entries are independent of any input, so implementations use pre-calculated forms if enough memory is available. It is possible to implement versions of AES that compute the S-box values during operation, trading flash for cost.

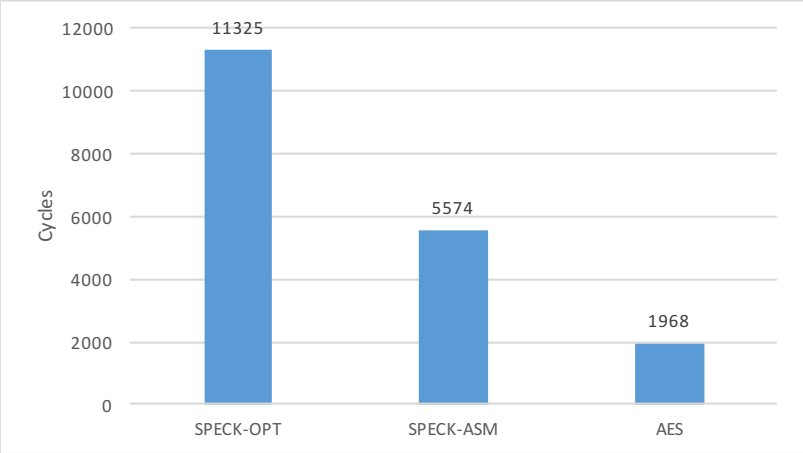
Results using SIV mode (on a 16-byte payload with a 16-byte header)						
Cipher	**Flash** (Bytes)	**RAM** (Bytes)	**Cost** (cycles/byte)	**Energy** (μJ/byte)	**Rank 1** (W_{cost})	**Rank 2** (W_{RAM})
I: Encryption and MAC creation (no key schedule generation)						
`SPECK-ASM`	5256	407	**3852**	**7.3**	**4.3**	**2.3**
`SPECK-OPT`	**4862**	410	6055	26.5	2.9	1.6
`AES`	7161	507	6646	32.5	1.8	1.0
`SPECK-EEPROM`	6196	**405**	9236	31.1	1.5	0.8
`SPECK-OG`	4925	413	835412	3237.6	0.0	0.0
II: Decryption and MAC verification (no key schedule generation)						
`SPECK-ASM`	5212	426	**4784**	**7.7**	**3.4**	**1.9**
`SPECK-OPT`	**4892**	429	6176	27.1	2.8	1.5
`AES`	7195	526	6759	32.49	1.8	1.0
`SPECK-EEPROM`	6198	**386**	9358	31.9	1.5	0.8
`SPECK-OG`	5007	431	835415.44	3238	0.0	0.0
III: Encryption and Decryption (including MAC steps and key schedule generation)						
`SPECK-ASM`	6408	1112	**10206**	**15.2**	**1.1**	**0.6**
`SPECK-OPT`	**5992**	1018	13704	57.8	0.9	0.5
`AES`	8095	941	13649	78.9	0.7	0.4
`SPECK-EEPROM`	7270	**520**	1758591	67.8	0.0	0.0
`SPECK-OG`	6829	1317	1815879	6887	0.0	0.0

Table 6.2: Cipher performance evaluation using SIV mode

`AES`'s decrypt operation is much more expensive than its encrypt operation—twice as expensive in terms of cost (see Figure 6-2a)—making it a less attractive candidate if decryption is required. STMicroelectronics (ST), a multinational maker of embedded hardware and software, shows similar speed differences with their AES implementations on one of their platforms [11]. Two of the key points of ST's explanation are:

- In AES block encryption, the `MixColumns` operation is easier to compute than `InvMixColumns` (used for decryption) because it uses smaller coefficients (hardware implementations can achieve the same number of cycles).
- And, in encryption, subkeys are used in the order they are produced, while in decryption the order is reversed requiring preliminary work before decryption

Figure 6-4: Cycles required to generate the key schedule



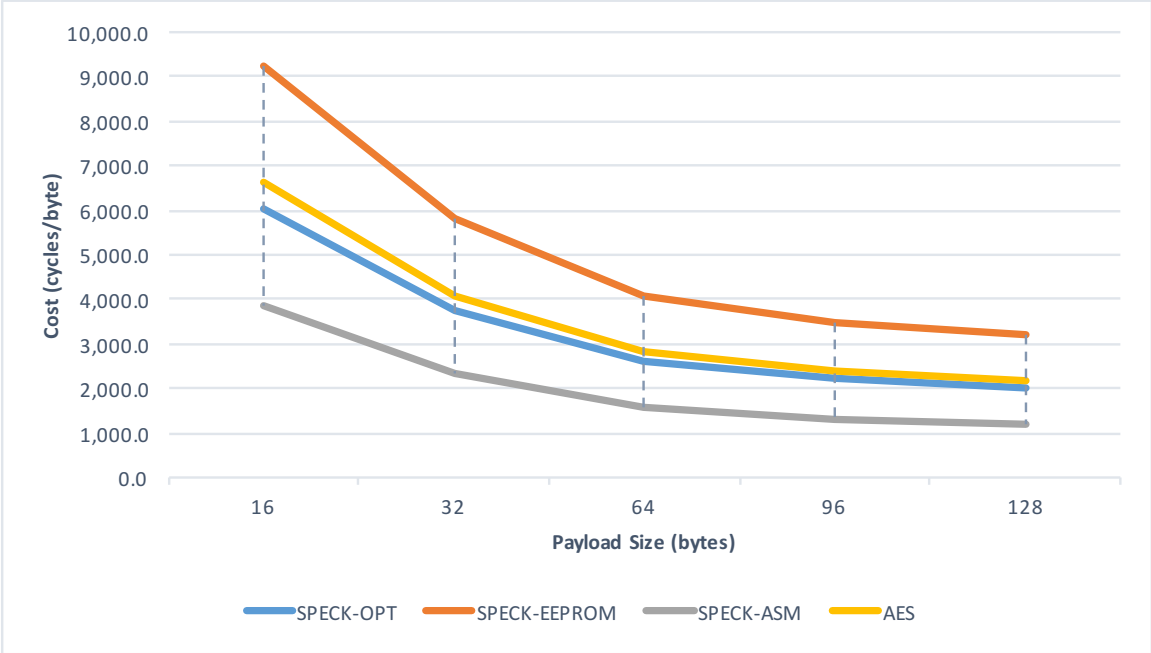
can begin.

Note, that with SIV mode, the underlying cipher's decryption operation is not required. In SIV mode, the payload is encrypted in Counter mode. In Counter mode, decryption uses the underlying block cipher's encryption operation. Additionally, the MAC creation and verification operations only require the cipher block encryption method. This explains essentially the same encryption and decryption performance in terms of cost for all the ciphers in SIV mode (see Figure 6-3a).

One area in which the `AES` cipher excels is key schedule generation. The chart in Figure 6-4 shows the number of cycles required to generate the key schedule for the top-3 ciphers. `SPECK-OG` and `SPECK-EEPROM` are omitted because they are not competitive. `SPECK-EEPROM` performs the worst because it has to write the key schedule into EEPROM. `AES` can generate its key schedule almost three times as fast as the nearest competitor, `SPECK-ASM`. Naturally, `AES` performs better than the other ciphers in benchmarks that do include the key schedule generation. While not the case for the RT-PSM system considered in this thesis, if keys must be replaced often, AES may be an attractive cipher option due to its fast key expansion.

A somewhat surprising result is that the closest `AES` competitor in terms of the *rank* metrics is `SPECK-EEPROM`. `SPECK-EEPROM`'s slow key schedule generation removes it from consideration in Scenario III (encryption, decryption, and key schedule generation), but in Scenarios I and II it is much more competitive. `SPECK-EEPROM` also

Figure 6-5: Cost of SIV encryption and MAC generation with respect to message size (using a 16-byte header)



competes closely with `SPECK-OPT` when `RAM` is more highly valued. Having the expanded key schedule in RAM greatly increases throughput at the expense of RAM use. `SPECK-EEPROM` performs poorly in Scenario III because it has a large key generation cost, since EEPROM writes are extremely slow—writing a single byte takes 3.3 ms to complete [6]. It's worth noting that if the program attempts to read the EEPROM memory before the writes have finished, the program will stall until the memory is ready. This can significantly slow down the encryption or decryption routines immediately following a key setup routine. If an application uses a Flexible-Key model, then storing the key schedule in EEPROM is probably not a good option[5].

I also benchmarked the ciphers in SIV mode on 32, 64, 96, 128 bytes of input (all using a 16-byte header) to simulate the effects of encrypting longer messages. As expected the relative ordering of cipher performance did not differ much in any of these cases from the 16-byte case. The results are shown in Figure 6-5. It's clear that cost goes down in SIV mode with respect to payload size. This is due to the constant

[5]Using a version of Speck that expands the key schedule in-line with the each encryption and decryption operation might work well.

amount of work needed to produce the message tag from the 16-byte message header. This indicates that depending on power constraints the payload size of streaming-type messages could be increased to reduce average encryption costs. If authentication, ciphertext integrity, and message header integrity are not required, tag generation overhead could be avoided altogether by just running the cipher in Counter mode.

As mentioned earlier, the reported cost metrics are the averages of 100 runs on random plaintext (or ciphertext) input.[6] The variance in the measurements was low: for all of the reported cost metrics, the 95% confidence interval has a width of 2 cycles/byte or less.

[6]In SIV mode the message header remained constant. In SIV mode and ECB mode, the keys remained the same.

Chapter 7

Optimizing and Securing the AUTOBAN Implementation

In this chapter, I cover some of details of my development process. First, Section 7.1 describes the steps I followed to improve the performance of my initial Speck-SIV implementation. Next, Section 7.2 details techniques I used to ensure the correctness and security of my implementation of Speck-SIV.

7.1 Performance Optimizations

In this section, I discuss how I improved my initial implementation of the underlying Speck cipher (`SPECK-OG`). This section aims to aid future developers in diagnosing and resolving performance bottlenecks in cryptographic libraries for embedded devices that support proof-of-concept applications like the AUTOBAN application described in Chapter 5.

If the aim is to produce a production quality cryptographic library, however, the following set of guidelines, similar to those discussed in [26], should be followed:

- The cryptographic library should be written in assembly in a single file and clearly commented. To this end, the developer should be familiar with the instruction set provided by her device. For instance, the ATtiny45 from Atmel's AVR family has a reduced instruction set that does not provide a hardware mul-

tiply. This means that the `SPECK-ASM` implementation used in the benchmark would not run on that device.

- The encryption and decryption processes should operate on text in memory—ideally in registers. The resulting ciphertext or plaintext should overwrite the passed in plaintext or ciphertext at the end of the process.
- The developer should understand the needs of her application in terms of flash, RAM, cost, and energy, and use a framework similar to the one presented in Section 6.1 to evaluate and select an optimal implementation. It is difficult to optimize for these metrics simultaneously and a developer should be open to making trade-offs.
- If necessary, in order to minimize data and memory usage, the key schedule should be precomputed and stored in flash or computed "on-the-fly" during encryption and decryption.
- If key flexibility is required, an interface for setting and securely erasing a key should be exposed.

Returning to the task of building cryptographic primitives to support proof-of-concept applications, as can be seen in the benchmark results in Table 6.2, `SPECK-OG` has multiple orders of magnitude worse performance than my optimized implementation, `SPECK-OPT`. This for the most part is due to a `ByteArray` class I used to hide the details of the bit arithmetic and logic on the 128-bit blocks. This choice was made to avoid confusion when initially implementing the algorithm. Speck uses the following operations on 128-bit words: bitwise `XOR`, bitwise `AND`, left circular shift, right circular shift, and modular addition (for more information on Speck see [13]). The initial `ByteArray` class provided extremely inefficient implementations of these operations. Further, the `ByteArray` object did not perform these operations in place. The `ByteArray` class also stored the underlying data in an array of `uint8_t`s in order to prevent extra work dealing with byte order differences between the network interface and the microcontroller. For all its inefficiencies, the `ByteArray` class allowed me to rapidly prototype a working implementation. The need to optimize `SPECK-OG` was born out of the AUTOBAN proof-of-concept application's streaming use case.

SPECK-OG did not allow for an acceptable level of throughput, as evidenced by the extremely large cost in Table 6.2. The application uses 128-Hz ECG data, and no reasonable message batch size I tried could transmit in real-time with SPECK-OG encrypting the batches.

The first step in producing a cipher with a lower cost was to understand where SPECK-OG was spending its cycles. For this I used Avrora in combination with GDB. This revealed that nearly all of the execution time was spent shifting. Speck requires circular shift operations by α and β amounts, where $\alpha = 8$ and $\beta = 3$ for the 128-bit block and 128-bit key version of the cipher. The ByteArray class included left and right shifts by 1. To shift by α or β, SPECK-OG looped over the shift-by-1 routines.[1] Creating more efficient shifting operations specific to α and β, yielded a 10x speed-up—better, but still 2 orders of magnitude away from the assembly version SPECK-ASM and still not capable of supporting ECG streaming.

The next step was actually a large leap. More comfortable with the Speck encryption and decryption routines, I eliminated the cipher's dependency on the ByteArray class altogether and implemented the encryption and decryption round routines as single-line macros. This provided a 500x speedup over SPECK-OG bringing it within a factor of 3 of SPECK-ASM. Instead of using bytes (uint8_ts) like the ByteArray class the macros used uint64_ts. On the ATmega328P shifting uint64_ts requires a function call, which explains some of the factor of 3 difference between this optimized version and SPECK-ASM.

The last step was to improve SIV mode which still made use of the ByteArray class. I rewrote the ByteArray class to operate in place and changed it to use uint64_ts as the underlying data type. To resolve the issue with the byte order, I opted for a less than ideal solution: using a flip operation on each of the two uint64_t values making up the 128-bit value when instantiating a ByteArray. A flip operation requires 6 shift operations, resulting in 12 total function calls when creating a ByteArray object. In spite of this, the new ByteArray class yielded a significant speedup over the existing one. In order to improve the implementation

[1] Circular shifts were built on top of these standard shift operations.

further, I could remove the need for the `ByteArray` class completely.

Optimizations complete, the fruits of my labor yielded a cipher competitive with a heavily optimized implementation of AES for ATmega328P, `AES`. To the original goal, the optimizations more than supported the throughput required for ECG streaming. Tables 6.1 and 6.2, highlight the results of the changes between `SPECK-OG` and `SPECK-OPT`.

7.2 Correctness and Security Tests

In this section, I detail the steps I followed to increase my confidence that the code I developed is correct and secure. It is well known that writing cryptographic code is full of perils. The simplicity of Speck and SIV mode made writing the code much simpler than other alternative ciphers and AEAD modes. For instance, Speck has no key-dependent values used for equality checking, branching, or table lookup. Additionally, my Speck and SIV implementations avoid using any dynamically allocated memory, reducing the chance for possible bugs. Nonetheless, I used the coding rules summarized in [9] to ensure security and to avoid common pitfalls.

Correctness. After finishing initial development, I tested my implementation of Speck with the test vectors given in [13]. I also encrypted random plaintext with my implementation and decrypted the results using other Speck implementations (I performed 1000 rounds of tests using the Speck implementations available through ArduinoLibs's cryptographic library [58]). I did the same in reverse to check decryption.

Security. To make sure that my implementation did not have any major security flaws, I ran it through two static evaluators for AVR builds: CppChecker (1.73) and Naggy (0.4.0) [10, 51]. The checkers revealed a few bad coding practices, which I resolved, but no major security issues. I also applied Valgrind's (3.11.0) Memcheck tool to check for memory errors (e.g., `valgrind --leak-check=yes --show-leak-kinds=all bin/speck-siv-tests`) and Clang's (3.4) AddressSanitizer to bounds

check on stack objects (e.g., compiled the test code with the `-fsanitize=address -mllvm -asan-stack` flags). While both of these last two methods operated on x86 builds, they each returned 0 errors and 0 warnings, giving me confidence in the correctness and security of my implementation. The checkers are not foolproof and the code needs to reviewed by people with experience writing secure libraries.

One issue I corrected in my SIV implementation was the `memcmp` used in the MAC verification step. Originally, the MAC verification step lacked a secure `memcmp` function [3]. In the MAC verification step, the tag generated from the decrypted ciphertext and header is compared against the tag provided. In order to be secure and avoid leaking information, the comparison between the tags should return in a constant amount of time regardless of what byte the comparison may fail on. The first implementation of `memcmp` that was used to compare the tags returned as soon as a difference was detected. This could potentially leak information to an attacker. In the AUTOBAN proof-of-concept implementation, however, it does not matter. When MAC verification fails on a message, the device will not respond, thus not leaking any information regardless of how long it takes to verify that the message is invalid.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 8

Conclusion and Future Work

Low-power computing devices and wireless transceivers have enabled real-time monitoring of physiological data through the creation of cheap, long-lasting components. Real-time monitoring of physiological data through an RT-PSM system can improve the health and readiness of both combat and noncombat military personnel, and can increase the situational awareness of squad leaders by providing information relevant for mission planning. The security of tactical RT-PSM systems warrants concern because a tactical RT-PSM system would necessarily exchange sensitive health information and data relevant to mission planning in a battlefield setting. Furthermore, allowing an adversary to inject messages would lead to confusion and to the deterioration of the system's availability, trustworthiness, and overall utility.

In this thesis, I have presented analysis techniques to aid the development of secure RT-PSM systems and demonstrated the application of these techniques by creating a secure proof-of-concept RT-PSM system implementation. In Chapter 3, I provided an adversarial model and methodology for analyzing threats against a generalized tactical RT-PSM system, which is detailed in Chapter 2. I considered the impact from loss of data confidentiality, integrity, and availability on the RT-PSM system from a variety of attacks that can be leveraged by adversaries ranging from script kiddies to nation states. Informed by this analysis and the derived top threats, in Chapter 4, I presented a set of security requirements and recommendations for enhancing the privacy and security of RT-PSM systems. Those requirements are:

- *Data integrity:* Data in transit between a hub and an EUD should not be corrupted or modified without the ability for detection.
- *Authentication and Authorization:* Components should be able to verify that a particular entity (or group of entities) created a message.
- *Data confidentiality:* Data should be readable only by authorized components of the RT-PSM system.
- *Data freshness:* Hubs and EUDs should be able to confirm that a message is recent and has not been sent before.
- *Software integrity:* RT-PSM systems should use code signing or other methods of checking software integrity to ensure that software placed on hubs and EUDs has not been tampered with to leak data or provide incorrect information to the squad leader.

The constrained nature of the components that make up a tactical RT-PSM system, in terms of power-budget, memory, and computational power, made the selection of mechanisms to fulfill these requirements a major challenge. Comparing alternatives, I devised a system centered around the Speck cipher and SIV mode (Speck-SIV), an efficient, symmetric-key based authenticated encryption mode, that achieves these requirements while supporting the system's original goal of improving the health of resilience of military personnel. Using Speck-SIV, in Chapter 5, I showed a functional proof-of-concept RT-PSM system capable of defending against the top passive and active threats identified in Chapter 3. I evaluated the performance of the cryptographic primitives in Chapter 6, showing that the performance of the Speck cipher is superior to the performance of the AES cipher with respect to RT-PSM systems. Lastly, in Chapter 7, I provided an overview of the development process and my methods of ensuring correctness and security of my implementation of Speck-SIV. While the proof-of-concept system can successfully defend against most of the top threats, several other areas require further research to build a truly secure RT-PSM system.

Encrypting Data at Rest. In the OBAN tactical RT-PSM system, hub devices store unencrypted sensor data on an SD cards for post-mission analysis. If a hub

device is captured, that data can easily be extracted. For future work, it would be helpful to explore options for encrypting data at rest. Unfortunately, any cryptographic keying material also needs to be stored in memory, so, even if the senor data is encrypted, the key could be used to decrypt it. Future work should also focus on how to securely erase a key when a device is lost or captured.

Software Integrity Checking. One of the remaining security requirements is software integrity checking. For the Android application running on the leader's display device, Android Studio [1] or the `jarsigner` command line tool [5] can be used for signing and verifying. For the code running on the hub devices, it is more difficult. In order to detect software integrity violations in the field, *Remote Attestation* is needed. Remote Attestation is a security service that would allow a trusted party to ascertain the software integrity of an untrusted hub device. Some minimalist strategies (and a good summary of existing techniques) is presented in [29].

Supporting Dynamic Groups and Re-keying. Some tactical RT-PSM system applications may require dynamic groups or transmit data sensitive enough to warrant changing the keying material in the field. Further research into key distribution is needed. There are some promising techniques that rely on the RT-PSM system features mentioned in Chapter 4.2, such a distance bounding authentication [50].

Transmission vs. Compression Trade-off. In SPINS, Perrig et al. note that most of the energy overhead from adding security protocols to sensor networks arises not from the additional computational costs but rather from the increased data transmission [44]. The Speck-SIV implementation only adds a 16-byte message tag. Nonetheless, it would be beneficial to understand the energy characteristics of increased data transmission. Specifically, it would be interesting to understand the energy and time tradeoffs between sending data normally and compressing the data before sending.

In summary, I analyzed threats to a tactical RT-PSM system and used those threats to derive requirements that informed the development of a secure proof-of-concept RT-PSM system called AUTOBAN. The hope is that the combination of analysis techniques, security building blocks, and development practices I applied

will foster the creation of secure RT-PSM systems. While this work was performed in the context of an RT-PSM system, many of the results generalize to the broader BAN and Internet of Things (IoT) spaces where sensitive information is collected and transmitted by networks of low-power, computationally-limited devices.

Bibliography

[1] Android studio: The official IDE for Android. [Online]. Available: https://developer.android.com/studio/index.html

[2] Authenticated encryption. [Online]. Available: https://en.wikipedia.org/wiki/Authenticated_encryption

[3] AVR Libc <string.h>: Strings. [Online]. Available: http://www.nongnu.org/avr-libc/user-manual/group__avr__string.html#ga4cd54dc9109f0d3da49d9c35e6441b61

[4] "IEEE p1363.2: Standard specifications for password-based public-key cryptographic techniques." [Online]. Available: http://grouper.ieee.org/groups/1363/passwdPK/draft.html

[5] jarsigner. [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jarsigner.html

[6] (2012, jan) Arduino - EEPROMWrite. [Online]. Available: https://www.arduino.cc/en/Reference/EEPROMWrite

[7] (2012) Tech notes: Lincoln open cryptographic key management architecture. https://www.ll.mit.edu/publications/technotes/TechNote_LOCKMA.pdf.

[8] *DoD, Resilient Military Systems and the Advanced Cyber Threat.* Defense Science Board Washington DC, 2013. [Online]. Available: http://www.acq.osd.mil/dsb/reports/ResilientMilitarySystems.CyberThreat.pdf

[9] (2014, dec) Coding rules. [Online]. Available: https://cryptocoding.net/index.php/Coding_rules

[10] (2016, jan) Cppcheck: A tool for static C/C++ code analysis. [Online]. Available: http://cppcheck.sourceforge.net

[11] (2016, jan) Um0586 user manual : STM32 cryptographic library. [Online]. Available: http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1743/PF259409

[12] S. Adams. (2016, feb) Dilbert: Sunday February 28, 2016. [Online]. Available: http://dilbert.com/strip/2016-02-28

[13] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of lightweight block ciphers," Cryptology ePrint Archive, Report 2013/404, 2013. [Online]. Available: http://eprint.iacr.org/

[14] ——, "The SIMON and SPECK block ciphers on AVR 8-bit microcontrollers," Cryptology ePrint Archive, Report 2014/947, 2014, http://eprint.iacr.org/.

[15] ——, "SIMON and SPECK: Block ciphers for the internet of things," Cryptology ePrint Archive, Report 2015/585, 2015.

[16] M. Bellare, P. Rogaway, and D. Wagner, "EAX: A conventional authenticated-encryption mode," Cryptology ePrint Archive, Report 2003/069, 2003, http://eprint.iacr.org/.

[17] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *J. Comput. Syst. Sci.*, vol. 61, no. 3, pp. 362–399, Dec. 2000. [Online]. Available: http://dx.doi.org/10.1006/jcss.1999.1694

[18] M. Bellare and P. Rogaway, *Introduction to Modern Cryptography*, 2005, ch. 3. [Online]. Available: http://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf

[19] J. Biddle, D. Brigada, A. Lapadula, M. Buller, and S. Mullen, "Oban: An open architecture prototype for a tactical body sensor network," in *Body Sensor Networks (BSN), 2013 IEEE International Conference on*, May 2013, pp. 1–6.

[20] D. A. Brock and P. D. Horoho, "Army medicine 2020 campaign plan," Tech. Rep., 2013. [Online]. Available: https://ameddciviliancorps.amedd.army.mil/CivilianCorps.aspx?ID=b2c81aa1-4d69-4219-a74d-90d5bcbfffbf

[21] W. Burleson, S. Clark, B. Ransford, and K. Fu, "Design challenges for secure implantable medical devices," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, June 2012, pp. 12–17.

[22] J. Daemen and V. Rijmen, "AES proposal: Rijndael," Tech. Rep., 1999.

[23] D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Groschdl, and A. Biryukov, "Triathlon of lightweight block ciphers for the internet of things," Cryptology ePrint Archive, Report 2015/209, 2015, http://eprint.iacr.org/.

[24] M. J. Dworkin, "SP 800-38b. recommendation for block cipher modes of operation: The CMAC mode for authentication," Gaithersburg, MD, United States, Tech. Rep., 2005.

[25] B. Earl. (2015, nov) Measuring memory usage. [Online]. Available: https://learn.adafruit.com/memories-of-an-arduino/measuring-free-memory

[26] T. Eisenbarth, Z. Gong, T. Güneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni, F.-X. Standaert, and L. van Oldeneel tot Oldenzeel, *Progress in Cryptology - AFRICACRYPT 2012: 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices, pp. 172–187. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31410-0_11

[27] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Des. Test*, vol. 24, no. 6, pp. 522–533, Nov. 2007. [Online]. Available: http://dx.doi.org/10.1109/MDT.2007.178

[28] D. J. Elaine Barker and M. Smid, "NIST special publication 800-56a: Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography," NIST, Tech. Rep., 2007. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-56Arev1_3-8-07.pdf

[29] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A minimalist approach to remote attestation," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 244:1–244:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616606.2616905

[30] P. C.-G. Francisco Martn-Fernndez and C. Caballero-Gil, "Authentication based on non-interactive zero-knowledge proofs for the internet of things," *Sensors*, jan 2016.

[31] K. Fu and J. Blum, "Controlling for cybersecurity risks of medical device software," *Commun. ACM*, vol. 56, no. 10, pp. 35–37, Oct. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508701

[32] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, Aug. 1986. [Online]. Available: http://doi.acm.org/10.1145/6490.6503

[33] V. Grosso, G. Leurent, F.-X. Standaert, and K. Varıcı, "LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations," in *Fast Software Encryption - FSE 2014*, Londres, United Kingdom, Mar. 2014. [Online]. Available: https://hal.inria.fr/hal-01093491

[34] J. Haigh and C. Landwehr, "Building code for medical device software security," 2015. [Online]. Available: http://cybersecurity.ieee.org/images/files/images/pdf/building-code-for-medica-device-software-security.pdf

[35] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and implantable

cardiac defibrillators: Software radio attacks and zero-power defenses," in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, ser. SP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 129–142. [Online]. Available: http://dx.doi.org/10.1109/SP.2008.31

[36] D. Harkins, "RFC 5297: Synthetic initialization vector (SIV) authenticated encryption using the advanced encryption standard (AES)," IETF, Tech. Rep., 2008. [Online]. Available: https://tools.ietf.org/html/rfc5297

[37] T. Iwata and K. Yasuda, *Fast Software Encryption: 16th International Workshop, FSE 2009 Leuven, Belgium, February 22-25, 2009 Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ch. HBS: A Single-Key Mode of Operation for Deterministic Authenticated Encryption, pp. 394–415. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03317-9_24

[38] ——, *Selected Areas in Cryptography: 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ch. BTM: A Single-Key, Inverse-Cipher-Free Mode for Deterministic Authenticated Encryption, pp. 313–330. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-05445-7_20

[39] C. F. Kerry and P. D. Gallagher, "FIPS PUB 186-4. digital signature standard," Gaithersburg, MD, United States, Tech. Rep., 2013.

[40] O. Landsiedel, K. Wehrle, and S. Gotz, "Accurate prediction of power consumption in sensor networks," in *Proceedings of the 2Nd IEEE Workshop on Embedded Networked Sensors*, ser. EmNets '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 37–44. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251990.1253399

[41] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, *Ambient Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ch. TinyOS: An Operating System for Sensor Networks, pp. 115–148. [Online]. Available: http://dx.doi.org/10.1007/3-540-27139-2_7

[42] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of AES," in *Advances in Cryptology EUROCRYPT 2011*, ser. Lecture Notes in Computer Science, K. Paterson, Ed. Springer Berlin Heidelberg, 2011, vol. 6632, pp. 69–88. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20465-4_6

[43] I. of Medicine (US) Committee on Metabolic Monitoring for Military Field Applications, "Monitoring metabolic status: Predicting decrements in physiological and cognitive performance." The National Academies, Tech. Rep., 2004.

[44] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: Security protocols for sensor networks," Secaucus, NJ, USA, pp. 521–534, Sep. 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1016598314198

[45] J. Radcliffe. (2011, aug) Hacking medical devices for fun and insulin: Breaking the human SCADA system. [Online]. Available: http://www.airspayce.com/mikem/arduino/RadioHead/

[46] R. L. Rivest, *Fast Software Encryption: Second International Workshop Leuven, Belgium, December 14–16, 1994 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, ch. The RC5 encryption algorithm, pp. 86–96. [Online]. Available: http://dx.doi.org/10.1007/3-540-60590-8_7

[47] M. Robshaw, "Stream ciphers: RSA laboratories technical report tr-701," RSA Laboratories 100 Marine Parkway Redwood City , CA 94065-1031, Tech. Rep., jul 1995. [Online]. Available: ftp://ftp.rsasecurity.com/pub/pdfs/tr701.pdf

[48] P. Rogaway and T. Shrimpton, "The SIV mode of operation for deterministic authenticated-encryption (key wrap) and misuse resistant nonce-based authenticated-encryption," 2007. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/siv/siv.pdf

[49] R. S. Ross, *SP 800-30 Rev 1, Guide for Conducting Risk Assessments*. Gaithersburg, MD, United States: National Institute of Standards & Technology, 2012.

[50] M. Rushanan, A. Rubin, D. Kune, and C. Swanson, "SoK: security and privacy in implantable medical devices and body area networks." Johns Hopkins University, Computer Science, Baltimore, MD, USA, 1466, 2014.

[51] S. K. Selvaraj. (2015, nov) Naggy: A live compiler diagnostics extension for atmel studio. [Online]. Available: https://github.com/saaadhu/naggy

[52] D. Singelée, S. Seys, L. Batina, and I. Verbauwhede, "The energy budget for wireless security: Extended version," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1029, 2015. [Online]. Available: http://eprint.iacr.org/2015/1029

[53] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 3rd ed. Pearson Education, 2002.

[54] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, *Selected Areas in Cryptography: 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ch. TWINE: A Lightweight Block Cipher for Multiple Platforms, pp. 339–354. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35999-6_22

[55] W. Tharion, A. Potter, C. Duhamel, A. Karis, M. Buller, and R. Hoyt, "Real-time physiological monitoring while encapsulated in personal protective

equipment," *Journal of Sport and Human Performance*, vol. 1, no. 4, 2013. [Online]. Available: https://journals.tdl.org/jhp/index.php/JHP/article/view/jshp.0030.2013

[56] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.*, April 2005, pp. 477–482.

[57] W. Trappe, R. Howard, and R. S. Moore, "Low-energy security: Limits and opportunities in the internet of things," *IEEE Security Privacy*, vol. 13, no. 1, pp. 14–21, Jan 2015.

[58] R. Weatherley. (2016, mar) ArduinoLibs: Cryptographic library. [Online]. Available: https://rweather.github.io/arduinolibs/crypto.html

[59] D. Wheeler and R. Needham, "TEA, a tiny encryption algorithm." Springer-Verlag, 1995, pp. 97–110.

[60] W. Wu and L. Zhang, *Applied Cryptography and Network Security: 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ch. LBlock: A Lightweight Block Cipher, pp. 327–344. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21554-4_19