Compression and Security Platform for the Testing of Wireless Sensor Network Nodes

Bojan Mihajlović

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

January 2008

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Engineering.

Abstract

This thesis considers the in-field testing of wireless sensor network (WSN) nodes as a means of providing increased network uptime. Such nodes operate with strict limits on energy, cost, computing power, and are prone to attack. While testing through software-based self-test (SBST) programs has advantages on WSN nodes, energy efficiency can be increased by compressing SBST programs before distributing them. Of several compression schemes considered for the task, the BSTW algorithm with Rice coding demonstrates node energy savings while providing a low memory footprint and good compression ratios. Test security is provided through a multi-layer scheme which uses a wireless transceiver chip to offload AES protocol cryptographic functions. The entire security scheme is found to add only approximately 2.5% to the energy needs of SBST program distribution. Finally, a testing protocol is created to combine high-level testing commands with the compression and security schemes.

Résumé

Cette thèse étudie le test sur le terrain de noeuds de résaux de senseurs sans-fil (RSSF) afin d'obtenir un temps de vie accru. De tels noeuds opeèrent avec des limites strictes en énergie, cot, puissance en travail, et sont sensibles d'être attaqués. Alors que l'auto-test basé en logiciel (ATBL) a ses avantages sur des noeuds RSSF, l'efficacité en énergie peut être améliorée en comprimant les programes ATBL avant de les distribuer. Une parmi plusieurs méthodes de compression étudiées pour la tâche, l'algorithme BSTW avec codage Rice démontre des économies d'énergie, tout en nécessitant une faible quantité de mémoire et de bons ratios de compression. La sécurité du test est assurée à travers une approche à plusieurs niveaux qui utilise une puce émetteur-récepteur sans-fil sur lequel décharger les fonctions cryptographiques AES. Toute l'approche concernant la sécurité ne rajoute qu'un 2.5 pourcent de besoin en énergie pour la distribution du programme ATBL. Finalement, le protocol de test est créé pour combiner des commandes de test de haut niveau avec des méthodes de compression et de sécurité.

Acknowledgments

I would like to thank the following people for their contributions to the completing of this thesis and to my successes thus far: My supervisors, Željko Žilić and Katarzyna Radecka, for their support, guidance, and advice, which has been instrumental in bringing this thesis to fruition. The reviewers, for their useful comments and their time investment. Rong Zhang, for the construction of the node and current measurement circuit used in my experiments. My parents, for their encouragement during not only the degree for which this thesis is written, but throughout my education. Vino Jeganathan, for her support throughout my studies and her willingness to share in my adventures. Nathaniel Azuelos, for motivating me throughout the thesis writing process, helping out when I was stuck, and translating the abstract to French. Jean-Samuel Chenard, for taking an interest and having the time to give me his opinions. Ahmed Abdel-Aaty, for always being willing to bounce ideas around with me. Sadok Aouini, for showing me the ways of *lab life*. My aunt and uncle, Jelena and Zlatko, whose relentless encouragement, commitment, patience, and support showed me what I am capable of, and allowed me to become the person I am today. The rest of my family and Blagojče Pogačevski, who have always believed in me and encouraged the pursuit of bigger and better things.

Што се хоће то се може – *народна пословица*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless networks of all types have seen increasing popularity in recent years. The technology that was once relegated to voice communication is now thriving as the new medium for transporting data. Applications such as wireless local area networking (WLAN), worldwide interoperability for microwave access (WiMAX), bluetooth, wireless USB, and cellular telephone systems such as 3G have seen widespread adoption in their respective markets. Of particular interest is the emergence of the wireless sensor network (WSN), which is poised to make it cost-effective to apply intelligent sensors to many residential, industrial, environmental, and military systems. In the residential sphere, its applications include smart temperature control, security systems, fire protection, lighting control, and home-automation. For example, a hotel operator may install a sensor node in every room to ensure that empty rooms are not being cooled with power-hungry air conditioning. Such an intelligent sensor network allows the cooling system to be connected to the check-in database. Technologically, control networks such as the example above have been possible for a long time now, but it is only with the emergence of WSNs that they have become cost-effective to implement.

1.1 Wireless Sensor Networks

1.1.1 Applications

A wireless sensor network is a system composed of embedded wireless nodes that cooperate on a common distributed application. An application typically falls under one of the

following categories [1]:

- Industrial control and monitoring
- Home automation and consumer electronics
- Security and military sensing
- Asset tracking and supply chain management
- Intelligent agriculture and environmental sensing
- Health monitoring

The nature of these applications is that of simple control-based communication or data gathering and requires very little data to be exchanged between nodes, usually at a rate measured in bytes or kilobytes per second. As a result, a node may be dormant for over 99.9% of its lifetime and will awaken only periodically to do useful work. Many WSN nodes can therefore be made to work for years on a single set of batteries if care is taken to minimize power consumption. With the exception of military applications, nodes must also be low cost if they are to be deployed in any appreciable quantity. A maintenance or repair mechanism need also be envisioned to ensure the availability of the network in the face of node failures.

If sensitive data is to be transferred over the network, some security system must be implemented to protect nodes from an attacker. Such a malicious adversary may want to, for example, use sensitive data to his advantage, take control of nodes in the network, remit false sensor readings, or eavesdrop on communications.

1.1.2 Network Protocols and Topology

The introduction of WSNs raised the question of the suitable protocols to use on network nodes of such low data-rate and power requirements. Existing protocols mentioned earlier, such as WLAN and Bluetooth, proved inadequate under such strict constraints. In response, the IEEE formed a new protocol and in 2003 published the first version of the 802.15.4 low-rate wireless personal area network (LR-WPAN) specification [2]. This is the basis for the high-level communication protocol known as ZigBee. Since 802.15.4 only defines the physical (PHY), medium access control (MAC), and data link layers (DLL) of the

OSI model, ZigBee has defined higher-level layers to form a full communication protocol for use on devices such as WSN nodes. The first version of the ZigBee protocol was defined in late 2004 and it too has undergone several updates since then. Among others, it works on the 2.4 GHz industrial, scientific and medical (ISM) radio band and the modulation and coding is built for reliability rather than speed. Speed options include 20 kbit/s, 40 kbit/s, and 250 kbit/s, and contained within the specification is a beacon-mode that allows a receiving device to sleep for most of its life, only waking up at predetermined intervals (beacons) to check if any data needs to be received.

Network topology is also of some importance in designing a WSN. Common topologies include:

- Ad-hoc – nodes exist in a physical configuration which is not predetermined and they must establish links and negotiate routing, security, and other parameters.
- Mesh – nodes are aware of link parameters and network topology, and speak to far-away nodes by sending packets across their immediate neighbours.
- Star – a designated basestation acts as a hub and communicates with all nodes, structuring communication and establishing all rules.

Ad-hoc networks are most complex, since nodes may not be stationary and may have their neighbours change over time. Security-related issues on such networks are an open topic of research, since there is no central authority that may be trusted. Mesh and star-based networks generally have a fixed topology and simpler design considerations. This thesis considers the use of a hybrid network with a simple yet practical design, whereby multiple star networks are connected together through their basestations. In this case physical topology is known and fixed, and security can be implemented by nodes having trust in only their basestation.

1.2 Security in Wireless Networks

The increased use of wireless technology in the recent past has highlighted the need to secure wireless devices of all types, including networks. The complicated task of eavesdropping on wired communication networks by physical interception is now trivial in the wireless networks steadily replacing them. An adversary need only be in proximity of a network

node to intercept all communication being sent and received. With the proliferation of WLAN technology came the use of security schemes such as wired equivalent privacy (WEP) to ensure the confidentiality of data transferred over wireless links. The protocol uses a cryptographic algorithm with a relatively small key-size (40-bit) to keep data private between the sender and receiver. As of its introduction in 1999, this key-size was the maximum that the United States government deemed acceptable for export outside of the US, by law. A change in US-policy in 2000 allowed for the export of "strong" cryptographic products to all but a few select countries [3], which was important for the evolution of more advanced security products. The circumvention of the WEP protocol became trivial after several vulnerabilities were discovered in its design in 2001 [4]. This highlighted the need for well-designed and tested security platforms based upon the more open US-policies. Testing could reasonably ensure that an encryption algorithm could withstand cryptanalysis. Increased key-sizes made brute-force attacks more difficult even with the availability of large amounts of computing power.

In the ensuing years after cryptographic export restrictions were lifted, one of the most prominent encryption algorithms to have been announced was called Advanced Encryption Standard (AES). This algorithm was officially adopted as a standard by the US National Institute of Science and Technology (NIST) in 2002 [5] in order to replace the older compromised Data Encryption Standard (DES) [6]. Its use is now widespread in securing data in communications and in other domains. It falls into the category of a block cipher, in which a source message is transformed into an encrypted message of the same length. The algorithm is also of the symmetric key form, which means the same fixed-length key is used to both encrypt and decrypt a source message.

The AES protocol has already proved useful in ensuring WLAN security, as it is a part of the IEEE 802.11i standard [7], also known as Wi-Fi Protected Access 2 (WPA2). In a simple scenario, a transmitting and receiving node can share a secret AES key. The transmitter will encrypt the payload portion of its packet with the key, replacing the unencrypted messages with an encrypted one of the same length. The packet is sent over the wireless interface, where the payload is unintelligible to any eavesdropper. Only the receiving node knows the shared key, and can decrypt the message with the same key that was used to encrypt it. The use of a single, relatively small shared key, and other characteristics discussed in Section 4.2 make the use of AES ideal for embedded computing. In this work it is used as a basis for a security framework for distributing data to WSN nodes.

1.3 Motivation and Thesis Contribution

When deploying WSN nodes in large quantities to cover large geographical areas, individual node reliability must be factored into the design to ensure the network is online and available. In the design of sensor networks, a node typically has several neighbours to which it may transfer its data. If several strategic nodes happen to fail, it may lead to the network becoming partitioned and effectively unusable to the application. Inevitably, low-cost WSN nodes will fail quicker than higher-priced devices, and regardless of cost all devices will fail at an accelerated rate in hostile environments. Therefore, a mechanism for uncovering failing and failed nodes would help identify those nodes which must be repaired or replaced before network availability suffers. In Chapter 2, traditional testing methods are shown to be inconsistent with the requirements of WSN nodes, and as a consequence a testing interface using software-based self-testing (SBST) is suggested. In an effort to further reduce WSN node power consumption, a new method of compressing SBST data is proposed in Chapter 3, which minimizes power-hungry communication over wireless links. In order to secure the new testing interface, a security scheme is suggested in Chapter 4 that uses encryption and secure design principles to protect against attacks. Finally, a testing infrastructure is proposed in Chapter 5 using the aforementioned compression and encryption systems to test WSN nodes in-field.

Chapter 2

Background

2.1 WSN Node Composition

The introduction of ultra-low power network protocols such as IEEE 802.15.4 [2] and its overlay protocol, Zigbee, has allowed wider use of a new class of devices. The new protocols enable devices to save energy otherwise expended in frequent communication, and are well-suited to WSNs. For the purposes of this thesis, a WSN is composed of sensor nodes and a basestation. The sensors nodes' main function is to collect data through environmental sensors, as well as manipulate actuators that control physical processes. Each sensor node may contain the following elements:

- embedded microcontroller unit (MCU)
- wireless transceiver
- antenna
- environmental sensors
- actuators
- a battery or other power supply

To minimize cost, components are often common off-the-shelf (COTS) parts assembled on a printed circuit board (PCB). In recent years, full-featured MCUs such as the Texas Instruments MSP430-series have become available that use as little as $3\,\mu W$ of power within

their sleep state. The low power consumption and low cost of these MCUs (presently on the order of $3–$6 each, in quantities of 1000) lends their use to low duty-cycle systems such as WSN nodes. Wireless transceivers may either be integrated on-chip with an MCU or be located in a separate piece on silicon that is placed onto the same PCB. A transceiver presents the highest power consumption of the system, but ZigBee and 802.15.4-compliant devices save power by the nature of their networking protocol. Antennas can be of the external type, but are more often dipole antennas printed onto the same PCB to reduce cost. Depending on the power consumption of the system, an on-board battery can last from several weeks to several years after deployment. Sensors nodes are also sometimes made to harvest energy from their environment to recharge the battery and increase system longevity.

2.2 Requirements

Having established the composition of a typical WSN node, the operational requirements for a successful deployment will be explored. Previous research has already established a general set of requirements in [1] and [8], but the definition here separates them in terms of network-level, node-level, and shared considerations. The following are general network-level requirements that a robust WSN should meet:

- Availability – The network should be online and available to the applications running on both the basestation and on the nodes themselves.
- Scalability – Networks with thousands of nodes may become commonplace in environmental monitoring or military applications. The system must scale to accommodate such uses.
- Self-Organization – Harsh operational environments leads to a high rate of node failures. A robust network must be able to circumvent failed nodes and organize to a new functional state.
- Timing – Real-time operation must be available to support applications where it is necessary.
- Data Aggregation – Sending data from individual nodes in large networks to a single

basestation may overwhelm available network bandwidth. The ability of some nodes to poll data from a cluster of their peers can alleviate this pressure.

Node-specific requirements include the following, but are not limited in scope to a node, as failure to meet them can have a network-level effect:

- Low Power Consumption – Nodes are often battery operated and must remain functional for months or years. Since manual replacement of batteries is often not possible, minimizing power consumption is critical in achieving a robust network.
- Low Cost – Since wireless sensor networks are potential replacements for present wired sensor networks, they must remain low-cost in order to make them a viable alternative.

The following is a consideration that should be taken into account at both the network and node-levels:

- Security – Nodes should be secured against adversaries, who must be unable to eavesdrop on, modify, or intercept communications within the network.

The following sub-sections will focus specifically on the availability and security requirements on WSNs by defining them both more thoroughly. Previous research that has brought different solutions to these requirements will also be presented.

2.2.1 Availability

To ensure that a WSN can operate for a long time, a reliability assessment must be performed on the failure rates of its individual nodes. As previously defined in general by [9], the Exponential Failure Law (EFL) can be used as a model for the reliability of nodes over time. The EFL states that component reliability with respect to time, *R(t)*, is governed by the probability density function of the exponential probability distribution seen in Equation 2.1, where λ is the mean component failure rate:

$$R(t) = exp(-\lambda t) \tag{2.1}$$

When the λt product is small, $R(t)$ can be approximated by Equation 2.2:

$$R(t) = 1 - \lambda t \tag{2.2}$$

To find Mean Time Between Failures (MTBF) of nodes, the integral of Equation 2.1 must be taken in the range $[0, \infty)$ with respect to time, t, as seen in Equation 2.3:

$$MTBF = \int_0^\infty exp(-\lambda t)dt = \frac{1}{\lambda} \tag{2.3}$$

Combining Equations 2.2 and 2.3 produces Equation 2.4, which relates reliability with the probability of node survival in a period of time t:

$$R(t) = 1 - \frac{t}{MTBF} \tag{2.4}$$

Even if redundancy techniques or highly reliable hardware are utilized, nodes can be deployed in harsh or even hostile environments that accelerate their failure. Of particular concern are failures that can disconnect the network and render a portion inaccessible. Based upon the same EFL as reliability, network availability can be defined as the probability that the entire network is available to use. This can be related to the Mean Time To Repair (MTTR) of one or more failed nodes, as seen in [10, 9], and here defined in Equation 2.5:

$$\begin{aligned} Availability &= \frac{System\ uptime}{System\ uptime + System\ downtime} \\ &= \frac{System\ uptime}{System\ uptime + (Number\ of\ failures)(MTTR)} \\ &= \frac{System\ uptime}{System\ uptime + (System\ uptime)(\lambda)(MTTR)} \\ &= \frac{1}{1 + (\lambda)(MTTR)} \\ &= \frac{MTBF}{MTBF + (MTTR)}, \quad since\ \lambda = \frac{1}{MTBF} \end{aligned} \tag{2.5}$$

The primary challenge is to maintain a working application in the face of node failures, while detecting and even predicting the failure of sensor nodes and their components. The probability that a network will be restored to full operation in a given amount of time after a failure has occurred is defined as network serviceability [8] or maintainability [9]. Following the analysis of reliability, the serviceability of a system, *S(t)*, can also be modeled on the probability density function of the exponential probability distribution, as seen in

Equation 2.6, where μ is the system repair rate:

$$\begin{aligned} S(t) &= 1 - exp(-\mu t) \\ &= 1 - exp\left(\frac{-t}{MTTR}\right), \quad since\ \mu = \frac{1}{MTTR} \end{aligned} \tag{2.6}$$

While Equations 2.1–2.6 allow a formal understanding of the metrics based upon simplified assumptions, in reality the calculation of such metrics is much more complex [1]. Distributed systems such as WSNs are subject to time-varying component connectivities due to factors impacting their wireless links [8]. Phenomena such as multipath fading, the "hidden terminal" problem [1], and electromagnetic interference (EMI) contribute to the complexity of calculating an important metric such as availability. For the purposes of this thesis, the concern is on overall system-level availability of a WSN application. So *perceived availability* is defined as the probability that a WSN application is functioning correctly over a period of time [8]. Since WSNs are also distributed in nature, the definition of reliability can be divided into *component-level* (local) and *process-level* (global) [11, 12]. While component-level reliability deals with the reliability of individual elements which comprise the distributed system, such as nodes, process-level reliability includes all processes, hardware components, and communication channels which constitute the WSN.

Component-level failures can affect process-level reliability, which in-turn affects perceived availability. Take the example of a functional mesh-connected WSN, where some nodes can maintain many connections and others as few as one. If the node deployment was not a careful one, there may be single nodes that act as gateways between smaller subnets within the WSN. A component-level failure of such a gateway node can cause it to fail, and the network to split. The process-level reliability of affected systems (such as communication channels) would decrease along with the perceived availability of the network. Failure detection and repair can take significantly longer due to channel unreliability and time overhead associated with retransmission timeouts [8]. However, the failure of nodes that have a system-level redundancy can be made to have little effect on process-level reliability as well as perceived availability. This is the case when packets can be routed around failed nodes in a mesh network through redundant communication links. The failure then has no impact on perceived availability. For further reading, a survey of the challenges and solutions in providing system-level reliability in WSNs can be found in [13].

Given that network availability and node reliability are interconnected, if node failures

could be detected then network down-time could be minimized. Periodic node testing is one way to ensure nodes are continually operating correctly. Since nodes can be repaired or replaced after they are found to be defective, testing can play an essential role in maintaining a high network availability. The following section will introduce various system-level testing options for digital systems and evaluate them against WSN requirements.

2.3 System-level Testing

Since WSN nodes are made to be deployed in inhospitable or even hostile environments, it is often impossible to obtain physical access to the devices in order to perform functional testing on them. Accelerated failures can also be attributed to the inhospitable environments in which WSNs might be used. Environmental conditions can compromise the reliability of node operation and by extension, the availability of the WSN itself. To achieve a high reliability and availability in a sensor network, node failures must be detected and/or averted during network operation. Thus, the in-field testing of individual nodes throughout their lifetimes is essential in predicting and detecting node failures and, in turn, improving network availability. Higher-priced systems, such as those in military applications, achieve enhanced reliability by using redundancy and highly reliable parts. However, such configurations have been shown to be unsuitable to power sensitive devices [11]. The cost and power consumption of such methods can quickly become prohibitively large on WSN nodes. It is clear that an alternative way of achieving the needed reliability and availability is needed that allows devices to retain a small power footprint and cost.

There are several methods of functional testing which can be used for system-level testing of nodes, including:

- Boundary Scan
- Hardware Built-in Self-Test (BIST)
- Software-based Self-Test (SBST)

Boundary scan methods require dedicated pins and larger, modified flip-flops to be physically incorporated onto a chip. Devices known as automatic testing equipment (ATE) are traditionally interfaced physically with the device-under-test (DUT). The ATE executes automatic test-pattern generation (ATPG) algorithms which create test vectors to stimulate

potential faults based upon some fault model. The test vectors are shifted into the chip through the dedicated pins, the tests executed, and test responses shifted out. To overcome the need for physical connectivity, some research has been done in creating overlays which allow remote testing capability, as in the boundary scan that wirelessly interacts with the DUT in [8]. The process of shifting test vectors and responses can also consume a disproportionate amount of power for the testing of a sensor node. Using BIST resolves this issue by using dedicated hardware on-board the DUT to generate test vectors to output a single resulting test signature. However, hardware BIST is often not possible due to the performance, area, and energy overheads of dedicated test circuitry [14]. Since BIST is most often not included on sensor node components, creating a custom chip would also be relatively expensive when there are cheaper common off-the-shelf (COTS) parts available.

Recent work in software-based self-test (SBST) programs [15, 16, 17] offers a way to achieve high quality testing with a small performance overhead and no area penalty. The SBST programs utilize an existing MCU's instruction set to perform a self-test of all digital and mixed-signal components on a WSN node. Various test sequences are run through the system bus, peripherals, transceiver, and MCU itself in order to uncover faults based on some fault model. A summary of the combined test results, also known as a *test signature*, can then be generated. The signature can either be directly passed onto a basestation, or compared to a known-good result stored within the SBST program itself. The recent work in SBST can be attributed to its numerous advantages in certain systems over traditional methods, including:

- Testing while chip is running at full functional speed (at-speed testing) [18]
- Generation of deterministic test vectors from SBST program code
- Unique signature responses contained within SBST program code and compared by MCU [18]
- No need for expensive automatic test equipment (ATE)
- In-field testability
- Energy efficiency

This self-testing approach is considered as an inexpensive way of achieving reliability, availability, and serviceability in a WSN, and is discussed at length in [8]. There are other

compelling reasons for the use of SBST. Certain tests can determine if the hardware or software of a node has been tampered with by an intruder, in order to maintain WSN security requirements. The same interface is compatible with providing manufacturer testing, where tests are broadcast to multiple nodes simultaneously. This reduces the need for individual node testing using expensive automated testing equipment (ATE) or alternative on-board testing interfaces. Recent work in developing SBST programs for embedded processors includes the work of [16], which was used as the basis for an overall WSN node testing methodology proposed in [19, 15].

Since SBST testing respects the network and node requirements presented in Section 2.2, and due to its other advantages, it is used as a basis for the node testing protocol developed in Chapter 5. Such an infrastructure for distributing and executing tests, as well as for maintaining security during testing, was found to be lacking from the literature. As part of the test protocol that has been developed, SBST programs are dynamically loaded as needed onto nodes by the basestation, which also collects back test responses signaling a pass or fail condition. A general description of the protocol functionality can be seen in Figure 2.1.

2.4 Communications Security

The testing protocol of Chapter 5 outlines a method to distribute and execute SBST programs remotely. However, effective testing requires that complete node access is granted to SBST programs in order to uncover a maximum number of faults. This seemingly contradicts the requirement that a node be secured against a malicious attacker, by adding another medium through which a node can be compromised. Communications systems, including the wireless security protocols discussed in Section 1.2, mitigate similar vulnerabilities through the adoption of a security policy. Such a policy is needed if SBST programs are to be used for testing WSN nodes remotely. The following section outlines common requirements and techniques in establishing security in communications.

The goal of a security policy is to allow systems A and B to communicate over an insecure channel in the presence of an active adversary, such as one based upon the Dolev-Yao threat model [20]. This commonly used model assumes the abilities of the adversary extend to eavesdropping on communication, alteration of data in the channel, and impersonation of any party to the communication. A robust security policy must then meet the following

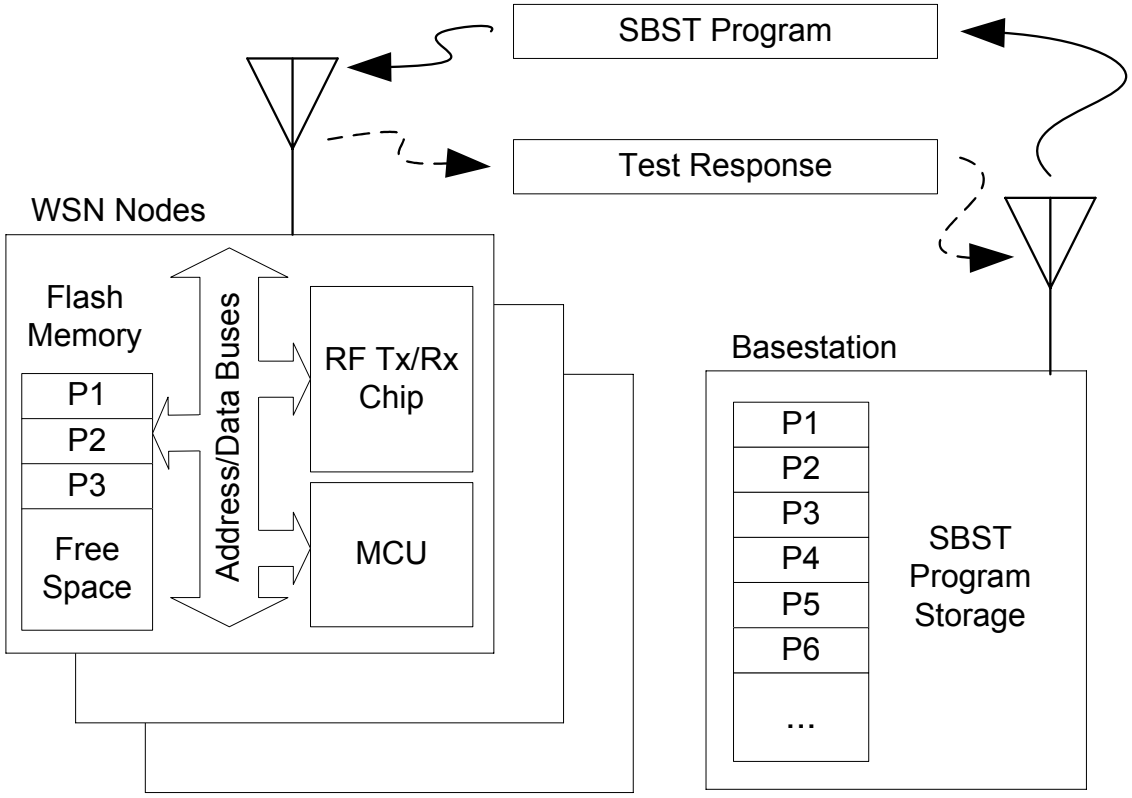


Fig. 2.1: General Description of WSN Node Testing Protocol

goals:

- Confidentiality – The data exchanged between A and B must be unintelligible to an eavesdropper, E.
- Data integrity – The data exchanged between A and B must remain unaltered in the presence of an adversary, M. It is assumed that M has the ability to control the channel by transmitting a higher-power signal over a part of the whole of the original transmission.
- Authentication – Systems A and B should be assured of each other's true identities, against an impostor I that pretends to be either system.

The most common method of meeting the stated security goals is by the use of a cryptographic system. Systems may consist of several discrete or interconnected components,

including:

1. *Cryptographic algorithms* the sender uses to encrypt data with a *key* before sending over the channel, and the receiver uses to decrypt data after receiving it.
2. *Hashing algorithms* that allows a fingerprint of each message to be generated and sent with the message.
3. *Digital signature schemes* that allows the sender to uniquely sign each message and the receiver to verify the signature.
4. *Key generation* methods to create the secret keys used in the cryptographic algorithm.
5. *Key management* mechanisms to distribute keys to, and revoke keys from the participants of the communication.

Components 1–4 have the property in common that they are based upon mathematically intractable problems. All ideally work in harmony to deliver system-level security that is impervious to most types of attack. The following subsections will explain these cryptographic system components in greater detail.

2.4.1 Cryptographic Algorithms

Cryptographic algorithms are procedures used to convert messages between an easily-readable *plaintext* form and an unintelligible *ciphertext* form. Algorithms perform the conversion using intractable mathematical problems or transformations such that only an entity holding the correct key may decipher the message. Cryptographic algorithms can generally be grouped into *symmetric key* and *public-private key* categories.

In symmetric key algorithms the communicating parties share a *secret key* that is only known to them. An algorithm consists of several types of mathematical transformations run in a sequence known as a *round*, with an algorithm recursively executing many rounds to complete its course. The same key is often used to perform both encryption and decryption of the message, with the receiving party performing the sender's mathematical transforms in reverse to recover the plaintext.

Symmetric key algorithms can be further divided into *stream ciphers* and *block ciphers*. The prior take each bit of the plaintext to be a separate message and operate independently

on them. They are well-suited to streams of data where there is no beginning or end to a data sequence, but are computationally intensive as they must repeat the process for each plaintext bit. Block ciphers operate on blocks of data at once, such that an n-bit plaintext is transformed into an n-bit ciphertext in a single invocation of the algorithm. Their practical characteristics include the ability to process greater volumes of data over stream ciphers, and the advantage of producing ciphertext of the same size as the original plaintext. Prominent symmetric block ciphers include DES and AES mentioned in Section 1.2.

In public-private key cryptography each party to the communication has both a *public key* and *private key*. The premise of these methods is that the one's public key is made known to anyone wishing to communicate with them. A sender uses the receiving party's public key to encrypt a message, which can only be decrypted by the receiver's private key. The advantages of public-private key cryptography are that the communicating parties need not share a secret key. Knowing a user's public key does not allow decryption of ciphertext sent to the user, so the user may share their public key with the world while remaining assured they are the only one with the ability to decrypt any ciphertext intended for them.

In general, public-private key algorithms are more computationally intensive than symmetric key algorithms. Their security is based upon the difficulty of solving a variant of either the *RSA problem* [21] or the *discrete logarithm problem* [22]. The nature of these problems necessitates a lengthier key size in well-known algorithms such as RSA and ElGamal over those of symmetric cryptography. The exception is a variant known as *Elliptic Curve Cryptography* (ECC) [23, 24] which uses key sizes comparable to symmetric techniques and offers slightly decreased complexity.

2.4.2 Hashing Algorithms

Cryptographic hashing algorithms allow a "fingerprint" of a message to be generated to achieve several security goals, including the verification of whether a message has been modified between sender and receiver. Such algorithms are effectively *one-way functions* that operate based upon the principle that fingerprints, or *hashes*, should be easily derived given a message, but reproducing the message based upon a given hash should be very difficult. Consider the following example, depicted visually in Figure 2.2:

1. A sender generates hash value H_1 by running message M_1 through a hash algorithm.
2. M_1 is concatenated with H_1 into a packet, encrypted with a cryptographic algorithm, and sent over an insecure channel.
3. The receiver decrypts the packet into received message M_2 and received hash value H_2.
4. The same hash algorithm is applied to M_2 which results in generated hash value H_3.
5. If the packet has been modified en route to the receiver, H_2 and H_3 will not be equal.

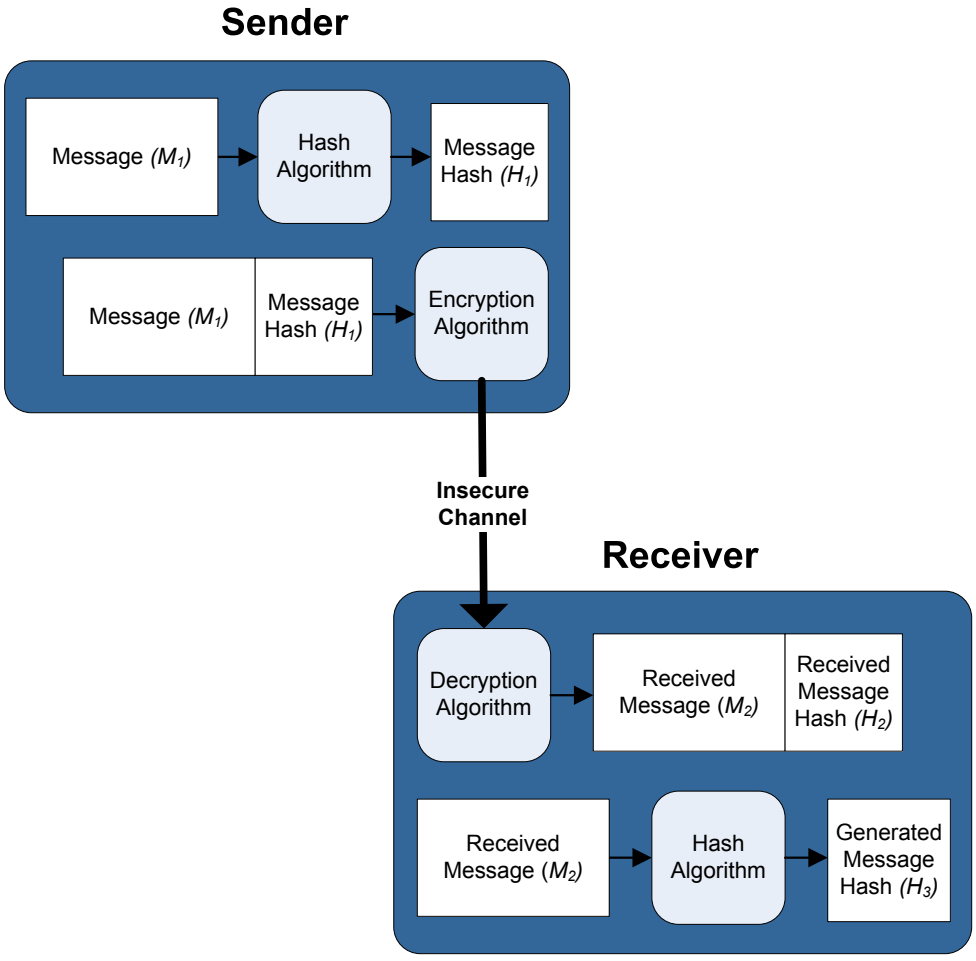


Fig. 2.2: Verifying Message Integrity through Hashing

This is an effective way of ensuring message integrity without incurring excessive overheads, since hashing algorithms can operate on any size of message to generate much smaller hash values. Hashing algorithms are also used in pseudo-random number generators, where a *secret seed* is used as input to the hash function. Some popular cryptographic hashing algorithms include the SHA and MD families of functions.

2.4.3 Digital Signature Schemes

Digital signatures are complementary to public-private key cryptography and ensure that the sender of a message can be verified. In public-private key encryption schemes, a receiver may use their private key to decrypt any message encrypted with their public key. Since all senders use the same key for encryption, there is no mechanism for verifying the identities of senders. In digital signature schemes, each communicating party also holds a set of public-private signature keys. For a message that is to be signed, a sender uses a hashing algorithm to derive its hash value. The hash value is then signed (equivalent to encrypted) with the private signature key and concatenated with the original message. The receiver can verify the signature (or decrypt) the hash value using the sender's public key. The received hash and a generated hash of the received message can be compared, as explained in Section 2.4.2. This ensures both message integrity and authenticates the sender of the message.

Some public-private key algorithms such as RSA have the ability to function in both encryption and digital signature capacities. While signing a message does not provide for confidentiality when sent over an insecure channel, if needed the message can always be encrypted after a signature is applied. Other well-known digital signature algorithms include DSA, part of the US government DSS standard, and Elliptic Curve DSA.

2.4.4 Key Management

The cryptographic algorithms introduced in Section 2.4.1 all involve the use of keys to meet security goals. To achieve the greatest measure of security and fully exploit the capabilities of the algorithms used, keys must be managed in the following ways:

1. Key Generation – When secret keys are generated, an attacker must not be able to determine or predict any of the contents of a key.

2. Key Establishment – Communicating parties should be able to securely agree on a key to use between them.

3. Key Distribution – When a secret key is generated remotely, it must be able to be securely distributed to all who require it.

4. Key Revocation – If an adversary is able to compromise a communication link: the security impact on other communication links should be minimal and when detected, the compromised entities should be excluded from future communication.

While the preceding sections have introduced cryptographic concepts that ensure security through mathematical complexity, these higher-level concepts are practical considerations in the functioning of cryptographic protocols. The following discussion will highlight the importance of each of the key management components listed above.

An ideal key generation method in symmetric key schemes occurs when each key is generated randomly and securely, giving the attacker no information to use in compromising a communication link. Practically, the generation of keys on devices that operate in deterministic ways, such as computers, creates the possibility that the process may be duplicated by an adversary. For this reason some key generation schemes rely on on-board hardware devices to generate random numbers. An alternative is to use a software-based pseudo-random number generator if its *seed* is kept secret and its output is fed into a hash algorithm. Other methods rely on remote key creation, where only one of the communicating entities or a trusted third-party is given the task. In the case of remotely-generated keys, the secrecy of the key rests upon the security of the key distribution method.

A key establishment scheme is responsible for delegating key creation to either a single party to the communication, or splitting the responsibility between many parties. The simplest method involves a single entity generating the secret key and using a key distribution scheme to transmit it to other parties that require it. However, a method such as Diffie-Hellman key exchange [25] allows two parties that have never communicated before to establish a secret key over an insecure channel. It relies on each party to generate a portion of the key, which is exchanged and applied to an algorithm. The distribution of the key is also handled as a consequence of the scheme.

Secret key distribution over insecure channels is needed if secure communication is to take place. When key generation occurs on a single entity and needs to be communicated to others, the key itself needs to be secured against eavesdropping. If communication has

occurred before between the same parties, the new key can be encrypted with the previous encryption key and distributed. If communication is being established for the first time, a method such as Diffie-Hellman key exchange will protect the key against eavesdropping.

The revocation of secret keys needs to be performed if a communication link has been compromised by an attacker. It involves informing other communicating parties of the security breach so they may avoid its compromised members or links. A communication scheme should be designed in such a way as to minimize the number of non-compromised parties excluded as the result of a breach. It follows that if a large number of parties are sharing a single key for communication, that a breach would compromise the entire group. If the original group could be divided into subgroups, a breach would be contained in any one of them, leaving the remainder unaffected.

2.4.5 Attacks

Secure cryptographic protocols must be designed with various threats in mind. Vulnerabilities must be considered at every level of a protocol, from the cryptographic algorithm itself to implementation-specific *side-channel attacks*. When considering attacks where both the sender and receiver are secure systems and the intruder only has access to an insecure communication channel, the Dolev-Yao threat model [20] is the de-facto standard at describing possible threats. It defines an *active saboteur* as "one who may impersonate another user and may alter or replay the message", whereas a *passive eavesdropper* "taps the communication line and tries to decipher the intercepted message". While the sender or receiver may not be perfectly secure systems, and attacks exist on other levels, threat modeling is useful in the analysis of security schemes against an established set of known threats.

2.5 Test Interface Security

Securing wireless communication has recently become a priority in many domains. Given the sensitive nature of some WSN data, security should be considered a priority in a practical deployment. In considering the testing of nodes through wireless links, a testing scheme should meet the following requirements:

1. Ensure security policy goals of Section 2.4 are met: confidentiality, data integrity, and authentication.

2. Respect of WSN node-specific and network-level requirements from Section 2.2.

Attention has recently been given to the issue of securing physical testing interfaces [26, 27, 28, 29, 30] which have historically been vulnerable to attack through access to hardware. Most of the work has been performed in securing against side-channel attacks using boundary scan interfaces, especially on devices containing sensitive intellectual property (IP) like cryptographic processors. Most of these systems use an obfuscatory reordering of I/O from the DUT in an effort to confuse the attacker. On the other hand, devices that perform testing through BIST are already inherently more secure since the same hardware contains both the test pattern generator and DUT, rendering the probing of intermediate signals much more difficult. Since SBST can be thought of as a software BIST, it shares the security advantage that its test pattern generator (SBST program code) and DUT reside on the same chip or board. It is then potentially more secure than regular boundary scan methods, without using the chip area needed for hardware-BIST. However, security issues surrounding the distribution and execution of SBST tests have thus far not been addressed in the literature. A security infrastructure developed as part of this thesis in Chapter 4 addresses this need.

2.6 Reducing Test Data Volume

As the speed of modern chips has increased, the speed of the testing interfaces to these devices has not followed suit. This, in addition to increasing design complexities have contributed to longer test times of devices seen as of late [31]. Compression of test data aims to reduce the volume of information transferred between a DUT and its tester. This is often done to reduce the overall time required to test a device, since the majority of test time is spent in the sending of test data and the receiving of a response. As a summary, the work thus far performed in test data compression has largely fallen into the following groups [32]:

1. Compression of fully-specified test vectors
2. Compression of incompletely-specified test vectors
3. Hardware Built-In Self-Test (BIST)
4. Compression/compaction of test responses

Methods 1, 2, and 4 require ATE to be connected to a DUT, where a remote interface could serve for in-field testing. For each test executed, a compressed test vector known as a *test cube* is generated by the ATE and transferred to the node where it is decompressed on-the-fly. The test is then executed and a response sent back to the ATE. In Method 1, the test cube is a directly compressed test vector generated by the ATPG algorithm, while Method 2 can employ many different schemes to further compress test vectors. In this method, the fact that most test vectors effectively contain many don't-care (X) bits is used to allow even greater compression. A type of *lossy compression* is applied, which is effective when only a few select bits of a test vector need to be exactly reproduced on the DUT. In Method 4, test response vectors can similarly be reduced in size. Compression can be performed on-board the DUT by the same algorithm as Method 1, or test responses can be compacted to signatures. These signatures are generated in much the same way as a simple hash, often by linear-feedback shift-register (LFSR) hardware.

All test communication for the aforementioned methods is done through a boundary scan interface. However, it has already been shown that the secret keys of cryptographic chips can be compromised using boundary scan attacks, and it is expected that boundary scan chains will become increasingly inaccessible on future production chips [29]. The test methods are also limited by the speed of the boundary scan chain, which can be significantly slower than a typical integrated circuit's operational frequency while incurring an area overhead. The exception is BIST, which can run at-speed and is improved through deterministic BIST methods [33], but is expensive in terms of physical device area and disallows the use of many COTS parts.

It is apparent that none of the compression methods mentioned can be applied to compressing the volume of test data contained in SBST programs. Since the programs either self-generate the vectors or contain them within the program code, a way to reduce test data volume is to compress the SBST programs themselves. An effective method is introduced in Chapter 3 to compress SBST programs, which are then stored in WSN node flash memory. Compared to methods 1, 2, and 4, tests execute at chip speed and subsequent executions of the same test do not require the SBST program to be transferred again.

2.7 Compression

Data compression is the process of removing redundancy from a data set in order to represent the original with a smaller set. A compression technique can be lossy, such that a decompressed data set need not be identical to the original when it is appropriate. Better compression can be achieved through the loss of information, such as parts of video or music when streamed over the Internet, where bandwidth is limited and continuous play is more important than a perfect reproduction of the original. This subset of lossy compression is known as *perceptive compression* [34]. Compression techniques can also be lossless, where the decompressed data set is identical to the original [34].

Entropy coding is a type of data compression where codes are assigned to a symbol set so that the lengths of the codes are inversely proportional to the probability that they will occur [35]. A symbol can be anything from a single bit to a long sequence of bits or other characters. Perfect compression would remove all redundancy from a data set by representing every symbol by exactly the number of bits needed to achieve a minimal length result. Every compression technique performs the following three operations [36]:

- Modeling
- Probability estimation
- Coding

Modeling performs an analysis on the structure of the data set, finding correlations between symbols and groups of symbols, or finding that each symbol is completely independent from the last. The less random the data set, the more redundancy can be extracted to yield better compression. In practice, the entropy of a data set is not usually measured in order to achieve perfect compression. Instead, *adaptive compression* techniques initially make assumptions as to the structure and correct those assumptions as they analyze the data they have compressed. Certain *nonadaptive* compression methods are useful only when the data type being compressed is known, as a single model is used and encoding cannot adapt to changes in entropy.

Based on modeling of the data set, probability estimation finds the probability of encountering each symbol. This is a type of statistics gathering which finds the perfect length code to represent each symbol with. Coding is the process of replacing each original symbol with the code representing that symbol. The original symbols are part of a *source*

alphabet, to which each code has an entry in the *codebook*. While the compressor develops the codebook in the process of compressing data, the decompressor needs access to the same codebook in order to decompress data. Adaptive compressors often do not need to communicate the codebook to the decompressor, as both start with the same assumptions and augment their codebooks based on the same rules as they process data.

In Chapter 3, several general-purpose compression algorithms are evaluated for use on SBST programs for the purposes of reducing the volume of test data transfered between the tester (basestation) and DUT (node). The memory requirements, complexities, and compression ratios of each algorithm are reported when tested against three real SBST programs.

Chapter 3

SBST Program Compression

In Chapter 5 a WSN node testing protocol is introduced toward the goal of increasing overall network availability, based upon the discussion in Section 2.2.1. The infrastructure defines how node testing is performed by SBST programs, which are transferred wirelessly between tester (basestation) and DUT (node) dynamically as needed. In Section 2.6 several methods are reviewed for reducing the volume of test data transferred, primarily to reduce node energy consumption. Since none of the established compression or compaction methods are suited for use on SBST programs, this chapter addresses the problem by:

1. Exploring the viability of compressing SBST program code
2. Considering the use of established general-purpose compression algorithms by comparing them with node-specific requirements
3. Evaluating compression performance between algorithms on real SBST programs
4. Measuring energy consumption in transferring compressed SBST programs compared to uncompressed ones, as per the test protocol

3.1 Characterization of SBST Program Code

The term *data compression* is defined in Section 2.7 as the process of removing redundancy from a data set. It is also said that the less random a data set, the more redundancy that can be extracted to yield better compression. If the compression of SBST programs is to be considered, the programs themselves should be characterized as being of low entropy.

Specifically, the frequency of occurrence of symbols within the program should be as far as possible from the *discrete uniform distribution*, where the probability of occurence of each symbol is the same. An analysis is therefore performed on three real SBST programs from [15] used in testing WSN nodes, denoted P_n in the following discussion. As a useful measure of entropy, a histogram of symbol frequencies is produced by inputting SBST program machine code into MATLAB 7.0. Code compilation is targeted at the Texas Instruments MSP430 MCU, a common WSN node device. In choosing a symbol size, a balance is struck between extracting maximum entropy and the ease of reading/writing each symbol. Byte boundaries are therefore respected, and a symbol size of 1 byte is chosen.

Figures 3.1, 3.2, and 3.3 show that the frequency of symbol occurrence is reasonably high in the measured SBST programs. The many peaks and valleys visually depict that the programs are far from the discrete uniform distribution and can thus be compressed. Similarities in frequently occurring bytes are also visible between the figures, which indicate that programs share similar characteristics. These similarities can be attributed to:

1. Instruction set structure

   The target MCU instruction set is composed of instructions for single-operand arithmetic, two-operand arithmetic, and conditional branching. Single-operand instructions share a 6-bit prefix such that the first byte of each instruction is *0x10–0x13* (bytes 16–19 in decimal). Likewise, the first byte of conditional branch instructions falls into the range *0x20–0x23* (bytes 32–35 in decimal). Programs will naturally use many of these instructions in the construction of SBST programs, and peaks can indeed be seen within these ranges in all three figures.

2. Commonly used instructions and operands

   Certain instructions are used more often in testing than others. For example, an efficient way to perform certain flash memory MARCH testing is to write to an array of data elements (MOV instruction), then read the array using an exclusive-OR (XOR instruction) between consecutive data elements. Certain operands are also more prevalent than others, such as passing *0x00* in the immediate addressing mode to test stuck-at-one faults in buses.

3. Shared code between programs

There is inherent overlap between SBST programs if they are to be executed independently of each other, as dictated by the test protocol in Chapter 5. Segments of code holding constants and performing hardware initialization must often be replicated, although future work may yield a more efficient solution where SBST programs can share this data.

The preceding characterization of three SBST programs suggests that there is an opportunity to compress such programs. In Section 3.2, several general-purpose compression algorithms are explored for use on SBST programs due to the lack of appropriate SBST-specific techniques in the literature. Algorithm attributes are compared with each other and against node-specific requirements, after which their performance and power consumption are quantified on a real WSN node in Section 3.4.

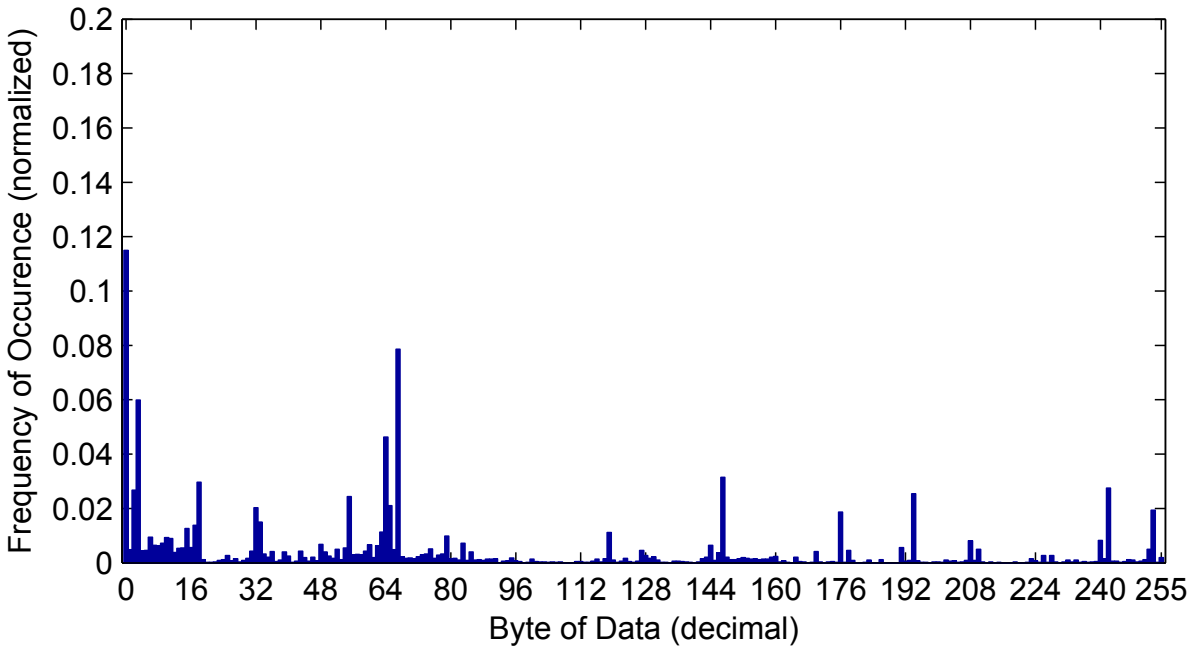


Fig. 3.1: Data Byte Frequency of SBST Program P_1

3.2 General-Purpose Compression Algorithms

There has been extensive work performed on entropy coding algorithms over the years, a summary of which can be seen in [37]. To uncover an effective method of compress-

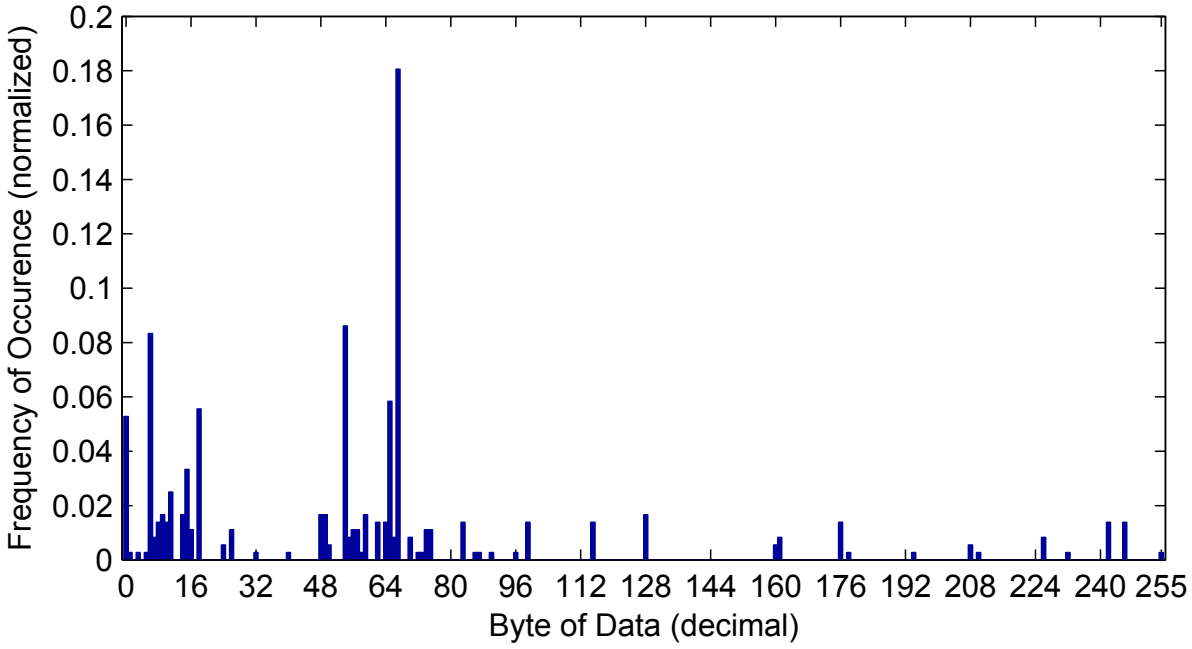


Fig. 3.2: Data Byte Frequency of SBST Program P_2

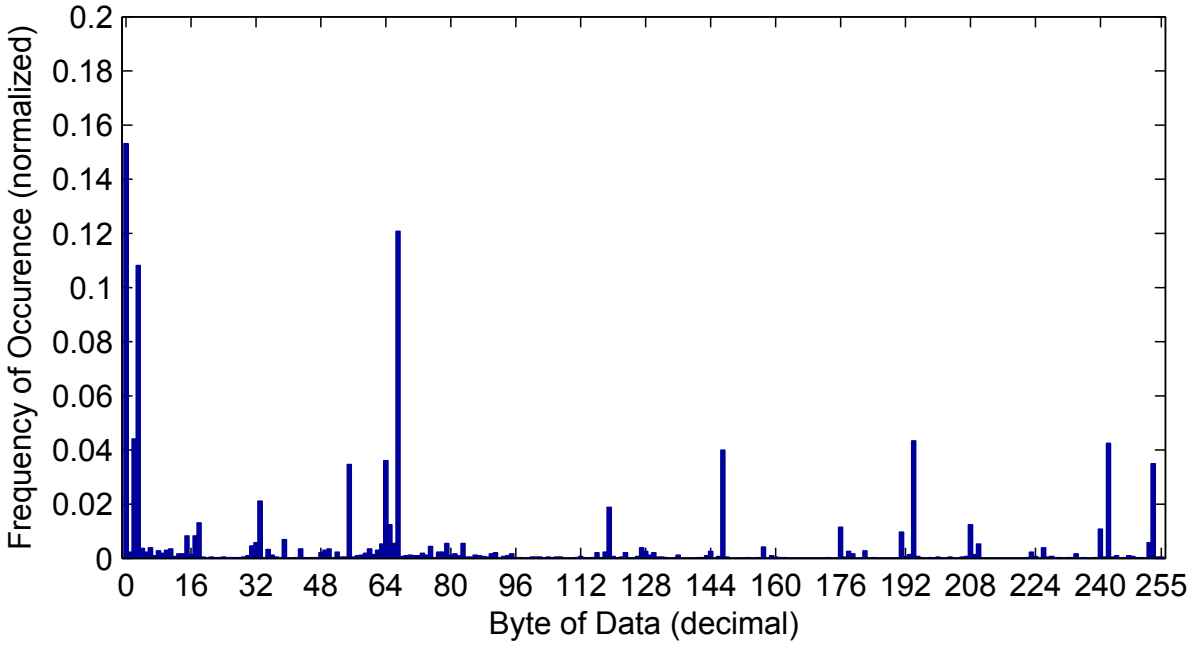


Fig. 3.3: Data Byte Frequency of SBST Program P_3

ing SBST programs, in the following section three groups of prominent algorithms are evaluated: Lempel-Ziv-Welch (LZW), Dynamic Huffman, and Bentley-Sleator-Tarjan-Wei (BSTW). The algorithms are compared against the following node-specific requirements seen in Section 2.2:

1. Low energy consumption

   By transferring compressed SBST programs instead of uncompressed ones, the power-hungry RF transceiver can have a smaller duty cycle and energy can be saved on-board a node. The node hardware must then be able to decompress the SBST programs without using excessive energy. The decompression algorithm should therefore be of low complexity and its implementation should be of low computational intensity.

2. Low cost

   Of the components in a typical node listed in Section 2.1, a disproportionate amount of cost is attributed to the MCU. A selection should therefore be carefully made in choosing an MCU meeting the minimum requirements of the WSN application. Since a typical application such as environmental monitoring requires modest resources, an MCU with small amounts of processing power, RAM, and flash memory is often enough. It is typical for an MCU to run at 8 MHz and contain 1 kB of RAM and 32 kB of flash memory. The strict limitations on RAM and flash memory must especially be taken into account when selecting a compression algorithm. This is important since the price difference between a common WSN node MCU with 5 kB of RAM compared to 1 kB can be almost **double** [38].

3.2.1 Algorithm LZW

The Lempel-Ziv-Welch (LZW) algorithm was published in 1984 [39], and is an improvement to the earlier LZ78 algorithm, which itself was an improvement on the LZ77 algorithm. After its discovery, the LZW algorithm became widely used in many applications for general-purpose compression, including the GIF and TIFF image standards. It displayed excellent performance due to its innovative approach in the construction of its codebook, by creating new symbols out of combinations of symbols already in the codebook [37]. Hampering an even greater adoption of the algorithm was (among others) the patent filed by Welch in 1983 [40], and the ensuing usage restriction on commercial products without licensing. All

patents covering the algorithm expired in 2004 and it can now be freely distributed and used.

The algorithm works by inputting a set of source symbols and scanning its codebook for the longest length match that is a prefix to those symbols. This means that the algorithm takes $O(n^2)$ time (where n is the set of source symbols), since both source symbol and codebook matching can vary in length. To take advantage of the full benefits of the LZW algorithm, the codebook size should be allowed to grow to several times that of the symbol set size. This allows more complex symbol combinations to be constructed for greater compression efficacy. Since the MCU and transceiver chip on-board a WSN node typically have register boundaries set at one or two-byte increments, filling an entire register upon every symbol received and making efficient use of these parts requires a minimum codebook size of 2^8 or 256 symbols. Using even this small symbol size, the LZW algorithm would quickly occupy the limited memory resources of the node. Even if the codebook is limited to $4S$ entries (where S denotes the number of unique symbols of the codebook), in a worst-case scenario the remaining $3S$ entries would require about 300 kB of memory. Nevertheless, this algorithm is considered in Section 3.2.4 because of its ubiquity and as a measure of the compression performance that can be achieved with larger amounts of memory.

3.2.2 Dynamic Huffman

The original Huffman coding algorithm [41] assures that a minimum-length code is developed across any particular data set. The algorithm works in two passes: gathering probability statistics from source data, and constructing a code based on those probabilities. The time taken is $O(n^2)$ since two passes must be made over the source symbol set. The codebook must also be separately communicated, since the receiver must know how to decompress the data set. Improvements to the original algorithm were subsequently made by Faller [42], Gallager [43], and Knuth [44], after which it came to be known as Algorithm FGK. Different improvements were later made by Vitter [45], and both sets of improvements are known as dynamic Huffman coding.

Dynamic Huffman coding is an adaptive compression scheme that omits the first pass of original Huffman coding. Instead of gathering statistics based upon the data set to be compressed, a codebook is generated dynamically as compression is taking place. As source symbols are mapped to codes, they are placed into a codebook with a hierarchical tree data structure. The observed frequency of source symbols is used to swap branches of the tree in

order to continually give frequently occurring symbols smaller codes. As with all adaptive methods, the performance of dynamic Huffman compression is initially poor but continues to improve as experience is built. One measure of any adaptive method performance is whether it is *asymptotically optimal*, which means that given enough garnered experience, the code lengths of the adaptive method shrink to those of a static method. In this case it has not yet been proven that dynamic Huffman techniques are asymptotically optimal [46], but practically their performance is close. The advantage these methods hold over original Huffman compression is that they only require $O(n)$ time, and that the codebook need not be transmitted since the receiver may reconstruct it by adapting his codebook lockstep with the sender.

While dynamic Huffman techniques offer a complexity less than that of LZW compression, the implications of maintaining a large hierarchical tree codebook are that it too requires significant memory resources. Each codebook *node* consists of the symbol frequency and the symbol itself, while the Huffman code of each symbol can be inferred from its position in the tree. Since nodes and branches of the tree are often swapped, in a practical implementation they must also be addressed with pointers which use memory. The result of tracking symbol frequency is that memory usage grows logarithmically as n increases. To limit memory usage, an upper limit to symbol frequency can be declared, in which case the algorithm will become static beyond that limit. Alternatively, the frequency tables can be deleted and statistics gathering restarted, but this negates the experience that the codebook has built.

3.2.3 Algorithm BSTW

Algorithm Bentley-Sleator-Tarjan-Wei (BSTW) [47] is an adaptive compression method simpler than dynamic Huffman and has the ability to outperform static Huffman in some cases [47]. As source symbols are input, the most frequent ones are given the shortest codes. In coding a source symbol, the algorithm outputs a code based on its location in the codebook, then uses a Move-to-Front (MTF) scheme to place that symbol at the front of the codebook (shown in Figure 3.4). This scheme ensures that symbols occurring frequently are efficiently encoded with the small codes, especially if they appear in bursts. It can be seen as an implicit form of *probability estimation*, as explained in Section 2.7. More importantly, symbol frequency is not explicitly tracked, as this is implicit in a symbol's position in the codebook and does not require additional memory.

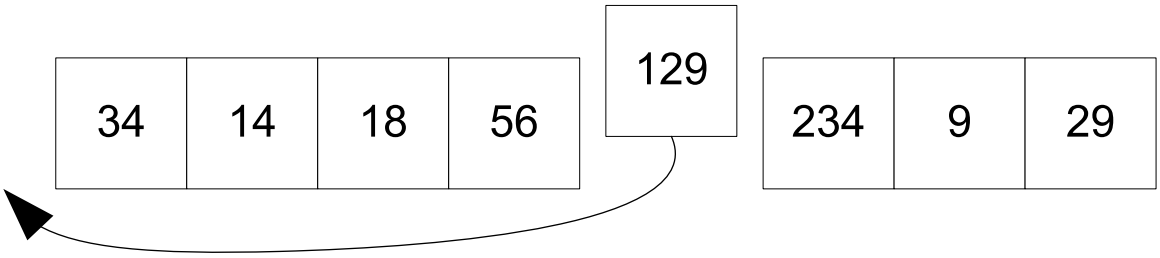
The compression algorithm itself requires only $O(n)$ time, since a single pass over the source data set is performed. Transmission of the codebook is also unnecessary for the same reasons as with dynamic Huffman techniques, although the algorithm is quite different from other adaptive methods. In BSTW the outputted codes are actually indexes to a changing codebook. The source data set is remapped from its original distribution into a data set resembling a *geometric distribution*, which can be imagined as source symbols being sorted into an order of descending frequency. An example of a reordered version of SBST program P_1 can be seen in Figure 3.5, whose original representation is in Figure 3.1. Also seen in Figure 3.5 is a superimposed geometric distribution which approximates the BSTW-reordered P_1.

Several implementation options for BSTW are possible, where the complexities of accessing the codebook data structure differ. In Table 3.1 two of these versions can be seen, with tradeoffs between memory usage and complexity. The pseudo-code for version B_2 of the algorithm can also be studied in Figure 3.6, which in addition to a codebook uses an equally-sized structure to cross-reference codebook data. A linear search of $O(n)$ is then saved if another S memory is available for use.

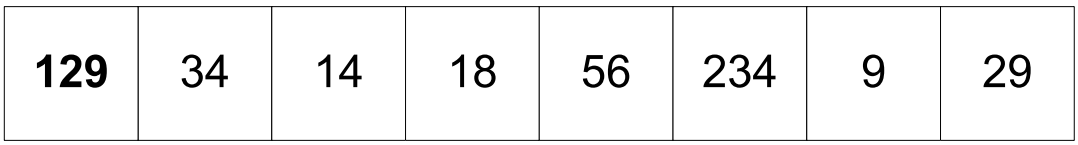
In a practical implementation reordered data takes up the same amount of space if reads and writes are performed within byte boundaries. If byte boundaries are broken, the symbols become indistinguishable from one another, or *ambiguous*. To solve this problem, BSTW codes must be re-encoded with a coding scheme such as universal codes or Golomb-Rice codes. Such coding schemes give optimal or near-optimal results to data sets following geometric distributions, and can be *unambiguously* decoded. In Section 3.2.4 the three groups of compression algorithms that have been presented will be compared to each other to find a promising match for use on SBST programs. Algorithmic complexity and memory requirements will be compared in order to highlight the differences between compression methods.

Table 3.1: Complexity Differences of two BSTW implementations

BSTW Version	Complexity over Source Data	Complexity in Data Structure	Memory Usage
B_1	$O(n)$	$O(n^2)$	S
B_2	$O(n)$	$O(n)$	$2S$



(a) Byte 129 encountered, whose code is '4' since it is 5^{th} from the beginning of the codebook



(b) Symbol array after shift, where a subsequently encountered byte 129 would have the code '0'

Fig. 3.4: BSTW Move-to-Front Heuristic

3.2.4 Compression Algorithm Comparison

In evaluating a compression algorithm for its potential suitability, the metrics of algorithmic complexity and memory footprint can be used as direct measures of meeting some critical node-specific requirements, as per the discussion in Section 3.2. These metrics are therefore compared in Figure 3.2 between the three aforementioned algorithms, where n represents the number of symbols processed, and S denotes the number of unique symbols of the codebook.

All of the compared algorithms fall into the category of adaptive compression schemes, while their adaptation methods differ. Algorithm LZW adapts by combining previous symbols to form new ones in an effort to improve its *data modeling*. This behavior is the most complex of all methods compared, and is shown in the greater orders of complexity and memory usage compared to the other algorithms listed. Neither dynamic Huffman nor BSTW methods track inter-symbol relationships to perform data modeling like algorithm LZW, but consequently both offer $O(n)$ complexity and lesser order memory usage. Dynamic Huffman methods perform adaptation by tracking symbol frequency and reshuffling their codebook to improve probability estimation, while algorithm BSTW uses the MTF heuristic to implicitly track symbol frequency in an effort to achieve the same goal. The difference between them is that dynamic Huffman methods require more memory to man-

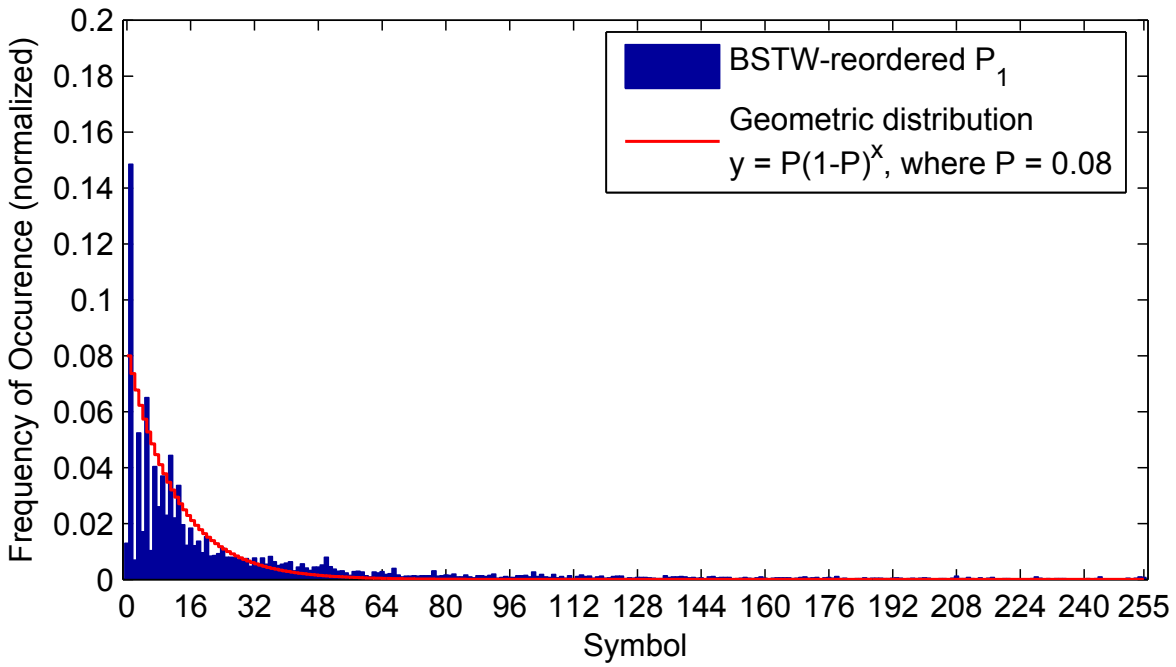


Fig. 3.5: Data Byte Frequency of BSTW-reordered SBST Program P_1 vs. Geometric Distribution $y = P(1-P)^x$ (where $P = 0.08$)

```
// codebook stores symbols, indices stores indices of codebook, inverse-indexed by symbol

for ( i = 0, i < size_of_source, i++ )
    index = indices [current_symbol]
    destination = index                    // write index to destination
    if ( index ≠ 0 )
       Remove current_symbol from codebook and shift codebook to fill hole
       Fill indices with new indexes to codebook
       Place current_symbol at front of codebook and fill indices with its index
    endif
Increment source and destination by a symbol
endfor
```

Fig. 3.6: BSTW Algorithm Pseudo-code (Version B_2)

age their more complex codebook data structure. Memory usage in dynamic Huffman also depends upon n, the number of symbols processed, while in BSTW it is a fixed amount irrespective of n. To bound memory usage in implementations of dynamic Huffman methods, a maximum block size is usually declared, after which the frequency tables are wiped

and statistics gathering restarted.

Section 3.3 will explore coding schemes which need to be applied to some compression methods, including BSTW, for unambiguous decompression to be possible. The coding schemes are evaluated based upon the same requirements as the algorithm comparison: complexity and memory requirements.

Table 3.2: Compression Algorithm Comparison – Complexity and Memory Usage

Compression Algorithm	Complexity over Source Data	Codebook Memory Usage (worst case)
Lempel-Ziv-Welch (LZW)	$O(n^2)$	$(S-1) + \frac{n\,(n+1)}{2}$
Dynamic Huffman *(Algorithm FGK)*	$O(n)$	$(S-1)\ log_2(n) + S + Pointers$
Algorithm BSTW	$O(n)$	$2S$ *(See Table 3.1)*

3.3 Static Coding

The compression algorithms explored in Section 3.2.4 did not make practical considerations as to how compressed data should be stored. In *binary coding*, symbols are stored along word boundaries regardless of how many bits within the word are unused. If the binary coding scheme is created to express any number between 0 and $n-1$, then $log2n$ bits must be used for any code in that range, regardless of the number of leading zeros prefixing a codeword. This is necessary so that the decoder knows where the boundaries lie between codewords, and is inefficient as a form of compression. In reality, compressed data is stored into a stream of sequential bits in memory or another storage medium. If binary coded data were to be stored in this way, it would not be deterministically recoverable. When one symbol is a prefix (or subset) of another symbol, decoding the stream leads to the possibility of multiple results. Of course, adding inter-symbol delimiters would disambiguate the codes, but the excessive overhead would give poor compression performance. The solution is to use a set of static codes which contain intrinsic delimiter bits and can thus be unambiguously decoded. Codes such as these contain the *prefix property*, such that no code is a prefix of any other code in the dataset.

Static coding techniques are effectively a simplified form of compression that either does not make use of probability estimation or makes only a small provision for it. Generally, one set of static codes is optimally suited to one particular probability density function (PDF). Since the output dataset of an algorithm such as BSTW has been shown in Figure 3.5 to approximately conform to geometric distribution, a static code suited to a geometric distribution could be used to express it. The result would minimize the delimiter overhead described earlier, while ensuring decoding would be deterministic. In the following discussion several types of coding are evaluated for complexity and memory requirements, as per the requirements discussed in Section 3.2.

3.3.1 Universal Codes

Universal codes are often used in conjunction with adaptive schemes because of the benefits of their prefix property, as discussed in Section 3.3. These codes map positive binary-coded integers into variable-length binary codewords. If the source symbol set is remapped in descending order of frequency (*monotonically decreasing*), such as BSTW-remapped Figure 3.5 is close to being, the advantage of using them includes the property that the resulting codes will be within a constant factor of the optimal code [37].

Asymptotically optimal universal codes include Elias-δ and Fibonacci codes, while Elias-γ codes are not [37]. All three codes include intrinsic inter-symbol delimiters, and a comparison between the mapping of their symbols can be seen in Table 3.3. From the code lengths, it is evident that each code would optimally encode a different symbol probability distribution. Elias code lengths increase at a faster rate than Fibonacci codes, and are better suited to geometric distributions whose probabilities are concentrated in the smaller symbols. Elias-γ codes are more difficult to compute than Elias-δ, but outperform them for small symbol sets. Errors in an Elias-γ codes are also often not recoverable, while Fibonacci codes are much more robust. Since each Fibonacci code ends in the suffix '11', a 1-bit error can at most cause the loss of 3 symbols before the decoder is resynchronized. Each of the codes is also generated according to algorithms of differing complexity, but Fibonacci codes are especially difficult to develop as the most popular method is recursive. This property would likely preclude Fibonacci codes from being generated as needed for *on-the-fly* decoding, and an implementation would need to store the codes in system memory for lookup purposes.

3.3.2 Golomb-Rice Coding

Golomb coding [48] is a form of static coding that allows for a degree of probability estimation to accommodate different source symbol PDFs. The original Golomb coding allows flexibility in selecting from many possible code sets, while the special-case version known as Rice coding [49] has greater code set restrictions, but is more useful in computing.

Both versions of the code are generated by dividing a source symbol by a *divisor*, where the quotient of the division operation is represented in unary coding and the remainder in binary coding. Unary and Rice codes for some symbols can be seen in Table 3.3, while an example of a typical Rice encoding can be seen in Figure 3.7. Generated codes have the property that the last bit of their quotient acts as a delimiter to separate it from the remainder, which disambiguates decoding of the code. Rice codes differ from Golomb codes in that their divisor setting is restricted to powers of 2. This restriction ensures that the division operation can be performed on an MCU by simple bit-shift instructions, while the remainder is a logical bit-mask with the original symbol. Such a simple implementation requires a minimum of memory and allows codes to be generated deterministically on-the-fly as source symbols are processed. Compared to the generation algorithms of deterministic universal codes reviewed in Section 3.3.1, Rice coding is potentially of equal or less complexity. Other promising advantages of using Rice coding include the ability to vary the divisor setting between datasets without penalty, making the coding scheme more flexible than most other static codes.

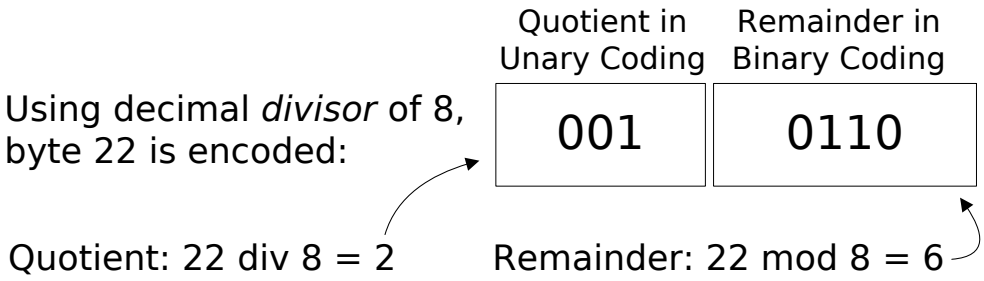


Fig. 3.7: Golomb-Rice Coding Example

Table 3.3: Comparison of symbol coding for various static codes

Symbol	Static Code				
	Elias-δ	Elias-γ	Fibonacci	Unary	Rice *
0	0	0	11	1	1 00
1	1000	100	0 11	01	1 01
2	1001	101	00 11	001	1 10
3	10100	11000	10 11	0001	1 11
4	10101	11001	000 11	00001	01 00
5	10110	11010	100 11	000001	01 01
6	10111	11011	010 11	0000001	01 10
7	11000000	1110000	0000 11	00000001	01 11

* with *divisor* setting of 4.

3.4 Experimental Results of SBST Compression

Three families of compression algorithms are qualitatively compared in Section 3.2.4 for use on SBST programs, specifically in their algorithmic complexity and memory footprints. Based upon these metrics, algorithm BSTW seems most promising for the intended application but is the only one that requires a static coding scheme to be applied in order to realize data compression. Section 3.3 explores potential static codes that can be used with algorithms such as BSTW, where Rice coding is found to contain several advantages that the others lack. In an effort to quantitatively analyze the differences between the algorithms presented in Section 3.2, the results of two experiments are presented in the subsequent sections.

The first experiment compares the compression ratios of the three algorithms (and their variations) on three real SBST programs. Results include an estimate of the minimum memory requirements for using each algorithm, based upon knowledge of their functioning and required data structures. The second experiment measures the energy required to receive compressed and uncompressed SBST programs on a real WSN node. Such results are necessary to evaluate the effectiveness of applying a compression scheme to SBST data.

3.4.1 Experimental Setup

In the experiment comparing compression ratios, three real SBST programs from [15, 19] (denoted P_n) are used with algorithm families LZW, Dynamic Huffman, and BSTW. The SBST programs are machine-code compiled for the Texas Instruments MSP430-family of microcontrollers, which can often be found in WSN nodes. Compression ratio results for algorithm LZW are collected using the UNIX utility *compress* v.4.2.4, while for the dynamic Huffman family of algorithms, an implementation of algorithm FGK by S. Toub [50] is used. To evaluate the performance of several static coding methods, two variations of algorithm BSTW are actually implemented in MATLAB 7.0. Functions for determining the length of various static codes are also implemented in MATLAB 7.0, which are applied to the BSTW-remapped data. As an example, the equation for determining the length of a Rice-encoded symbol is given in (3.1).

$$\left\lfloor \frac{symbol}{divisor} \right\rfloor + 1 + log_2(divisor) \tag{3.1}$$

The energy measurement experiment is performed with a real WSN node, known as the *WSN research platform*, which consists of a Texas Instruments MSP430F149 MCU and Chipcon CC2420 "engineering sample" ZigBee wireless transceiver chip, assembled on a custom printed-circuit board (PCB) with printed dipole antenna. A photograph of this node can be seen in Figure 3.8. The WSN research platform is connected to a custom current-measurement circuit, which can be seen in Figure 3.9, and allows energy consumption measurements to be made. Both the research platform and current-measurement circuit are developed in-house [15, 19]. The output of the current-measurement circuit includes a voltage that is measured across a known resistance, giving an instantaneous current reading. This allows instantaneous power to be found, which when plotted over time, can be integrated to find total node energy consumption. All current-measurement circuit outputs are analyzed in this way using an Agilent Infiniium 54830D 600 MHz mixed-signal oscilloscope. The energy contribution of only the transceiver is found by measuring the node energy consumption while the MCU is in a low-power sleep state and the transceiver is operating in *listen mode*. Since the current draw of the *listen mode* approximates the published current draw of *receive mode* of the CC2420 chip [51], it is used to isolate the transceiver component of total WSN node power consumption. Such an approximation is necessary since the transceiver would otherwise never be in *receive mode* with the MCU in

a low-power sleep state.

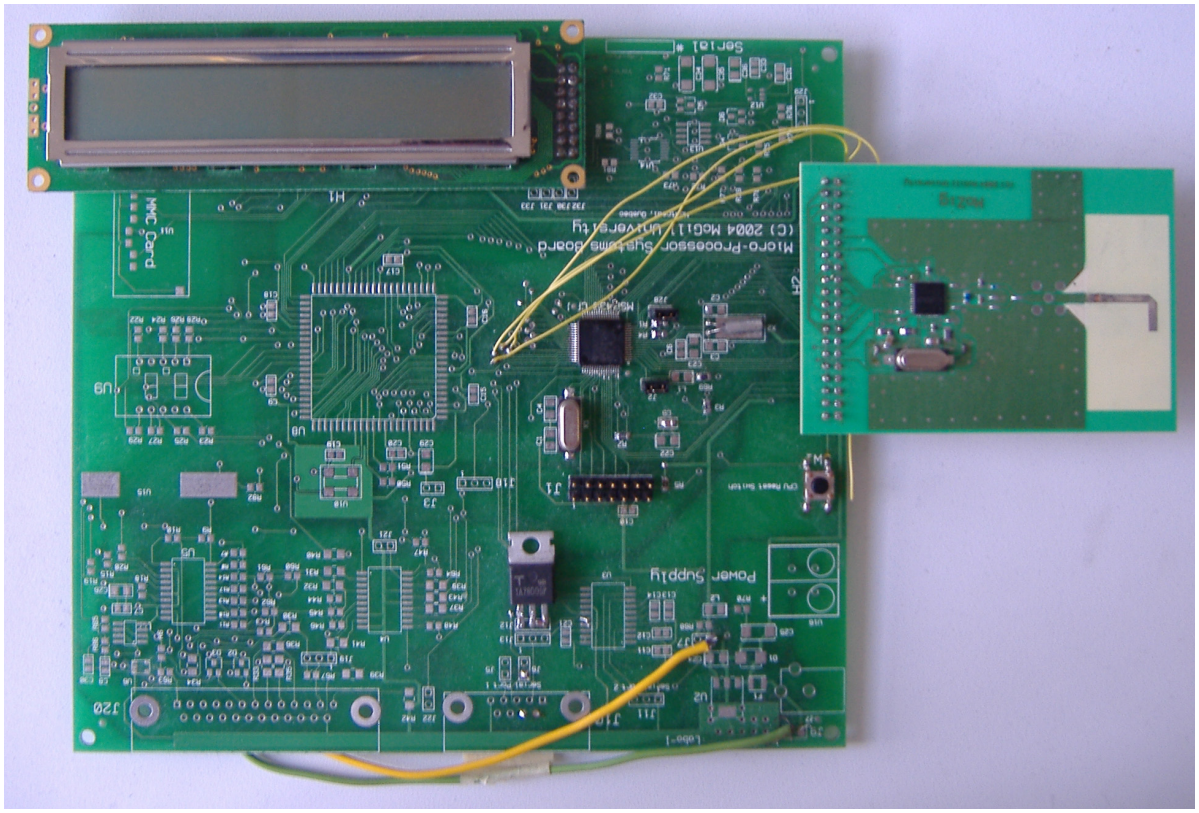


Fig. 3.8: Photograph of WSN Research Platform

3.4.2 Compression Ratio Comparison

Algorithm BSTW is introduced in Section 3.2.3 as solely using the MTF heuristic, but in this experiment an alternate heuristic denoted *swap* is also explored. In the MTF heuristic, when a new symbol is encountered it is moved to the front of the list, while using the swap heuristic it is exchanged with the element at the front of the list. A comparison of compression ratios and memory usage for the aforementioned algorithms can be seen in Table 3.4. It can be seen that algorithm LZW gives both the best compression ratios across all three SBST programs, as well as the greatest memory usage. Since results are collected using a compiled utility, the minimum memory requirements are estimated to be 8 kB, although the value is likely much higher. This very conservative estimate is enough to disqualify algorithm LZW from use on WSN nodes for requiring excessive memory

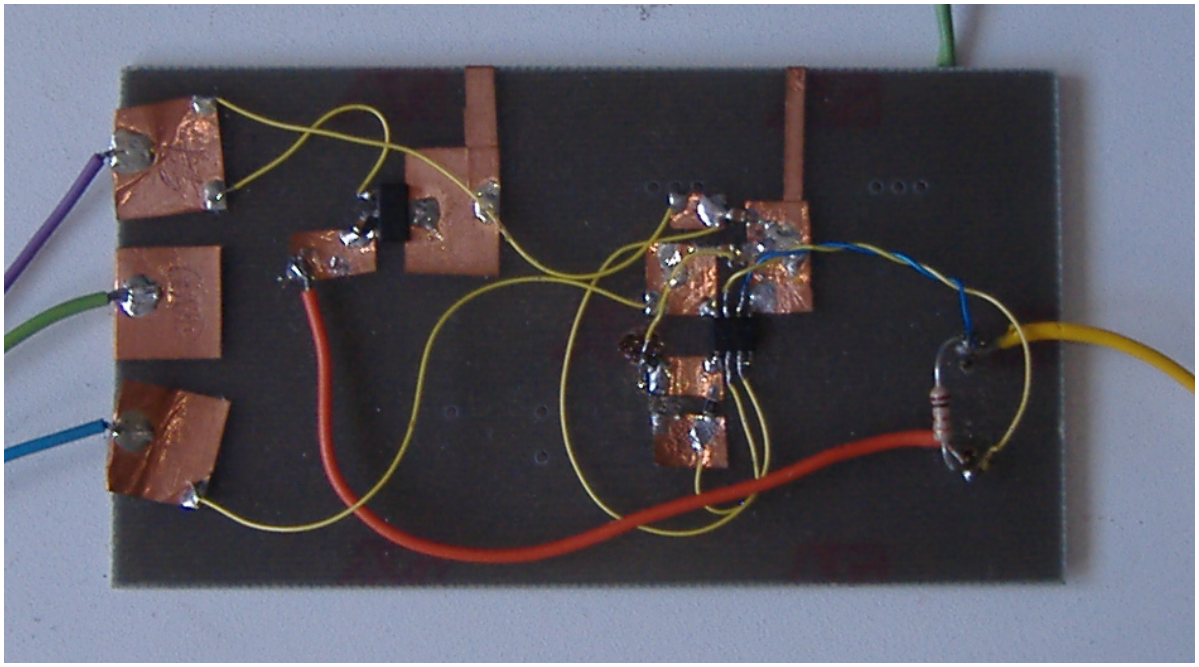


Fig. 3.9: Photograph of Current Measurement Circuit

resources, although its results are a useful benchmark for gauging the relative compression performance of the other algorithms. Programs P_1 and P_2 are compressed by LZW in the 30% range while P_3 sees a 55% reduction.

The memory requirements for algorithm FGK are estimated from the analysis in Table 3.2 for the same reasons as algorithm LZW, although it is found to require approximately 1.7 kB of memory. Achieved compression ratios for SBST programs P_1, P_2, and P_3 are 15.8%, 19.4%, and 27.5%, respectively. These compression ratios exceed those of BSTW (MTF) with Rice codes for divisor settings of 8 and 64, for all three SBST programs. Compared to BSTW (MTF) with universal codes, performance is better for P_1 and P_3 by a small margin.

Algorithm BSTW with static codes has the lowest memory requirements of those compared (see Table 3.2), at 0.5 kB. Performance of the algorithm with swap heuristic is disappointing across all static coding methods, with many cases experiencing an inflation instead of compression. Using heuristic MTF instead, the compression ratios are more encouraging. When combined with Rice coding with a divisor of 32, better performance over algorithm FGK is seen for P_1 and P_2, by a slim margin. Using a divisor of 16, the performance of algorithm FGK is exceeded by a slightly larger margin for P_1 and P_2. Even

though algorithm FGK gives better results for P_3 in all cases, the compression ratio of Rice coding with a divisor of 16 only lags behind by 3.6%, impressive given the over 3x memory savings.

Table 3.4: Achieved Compression Ratio vs. Memory Usage for three SBST Programs (denoted P_n)

Algorithm		Compression Ratio (%)			RAM Use (Bytes)
		P_1	P_2	P_3	
Uncompressed		0	0	0	0
Lempel-Ziv-Welch (LZW)		38.8	32.2	55.4	7936†
Dynamic Huffman – Algorithm FGK		15.8	19.4	27.5	1728‡
BSTW (MTF)	Rice Codes ($divisor = 8$)	7.7	16.7	18.4	512
	Rice Codes ($divisor = 16$)	18.8	22.5	23.9	
	Rice Codes ($divisor = 32$)	17.3	19.7	19.5	
	Rice Codes ($divisor = 64$)	9.9	10.3	10.7	
	Elias-γ Codes	8.6	24.2	19.2	
	Fibonacci Codes	15.6	23.9	22.2	
BSTW (swap)	Rice Codes ($divisor = 8$)	-31.9	-24.7	-29.6	512
	Rice Codes ($divisor = 16$)	-1.4	3.1	-0.6	
	Rice Codes ($divisor = 32$)	7.4	10.3	7.6	
	Rice Codes ($divisor = 64$)	5.3	7.8	5.3	
	Elias-γ Codes	-2.8	-14.7	5.1	
	Fibonacci Codes	8.3	0	13.4	

† codebook of 4096 strings, where the first 256 symbols are 1 byte, and the rest at least 2 bytes (very conservative estimate).
‡ assuming 256 symbols, 16 kB blocks, and 2 byte pointers.

3.4.3 Energy Expenditure Comparison

In Section 2.7 it is said that the reason for using compression on SBST programs is in an effort to reduce the volume of test data transferred between the tester (basestation)

and DUT (node). Reducing the volume of data received by the node directly leads to a reduction in node energy consumption since the transceiver presents the highest power consumption of the system. In this experiment the goal is to quantify the energy savings of transferring compressed SBST programs to a real WSN node from a basestation. Since algorithm BSTW and Rice coding (with a divisor of 16) is found to give a relatively good compression ratio and compatible with memory-limited WSN nodes, it is the algorithm used for this portion of the experiment. Both compressed and uncompressed versions of the same three SBST programs from Table 3.4 are transmitted, and the total energy consumption of the node is measured in receiving the programs.

When the transceiver is receiving data, the ratio of current draw between the MCU and transceiver is found to be approximately 1 to 4.5, from Table 3.5. In isolating the transceiver current-draw, the energy usage for receiving both compressed and uncompressed SBST programs can be seen in Table 3.6. The result is that the reception of compressed SBST programs over uncompressed ones yields an energy savings of 18.8%, 22.5%, and 23.9% for programs P_1, P_2, and P_3, respectively. It can be observed that these energy savings are directly proportional to the achieved compression ratios seen in Table 3.4.

Table 3.5: Contribution of current-draw for components when transceiver is receiving data

Component	Average Current (mA)
MCU operation	4.327
Transceiver operation	20.970

Table 3.6: Energy Usage for Wireless Reception of three SBST Programs (denoted P_n)

Function	Energy Usage (mJ)		
	P_1	P_2	P_3
Wireless Reception of Uncompressed SBST Program	14.727	0.779	9.420
Wireless Reception of Compressed SBST Program	11.955	0.603	7.167

3.5 Discussion

In the first experiment it can be seen that algorithm BSTW (MTF) and Rice coding with divisor 16 give excellent results for their memory requirements. Of the algorithms evaluated, only BSTW can be used on nodes with the smallest amounts of RAM. Surprising, however, is the result that algorithm BSTW combined with Rice codes with divisor 16 achieves greater compression than algorithm FGK for two of the three SBST programs. Nodes with greater amounts of RAM can consider this compression option since it could allow for a real-time operating system (RTOS) to perform decompression while other tasks run in the background. The low complexity and low memory requirements of the chosen compression scheme are consistent with the needs of WSN nodes.

The second experiment measures the energy savings from receiving compressed SBST program over uncompressed ones. It is shown that an appreciable energy savings is realized that is directly proportional to the compression ratio of the SBST program. In using compressed SBST programs, the node-specific requirements seen in Section 3.2 have been respected and even improved upon. A lower energy consumption can be realized without the need for more expensive node hardware, even when the energy savings in remitting compressed test responses is not considered.

This chapter explores several compression methods and combinations of static coding schemes for use on SBST programs. Compression ratio results of three real SBST programs identify the combination of a reordering and static coding scheme that is consistent with WSN node and network requirements. The reordering scheme is shown to use little RAM compared to similar algorithms, while the deterministic coding scheme likely requires no RAM. An appreciable energy savings is demonstrated on-board the WSN research platform node in transferring compressed SBST programs over uncompressed ones. This compression scheme is therefore incorporated into the testing protocol of Chapter 5 in order to save energy while providing the required WSN availability.

Chapter 4

Security Infrastructure

From the discussion in Section 2.2.1, it has been established that periodic WSN node testing is of paramount importance to maintaining network availability in the face of node failures. The node testing protocol of Chapter 5 presents a method for a basestation to distribute SBST programs to nodes, execute them remotely, and collect back test signatures. As is mentioned in Sections 1.2 and 2.5, attention has recently been given to security within both the wireless and digital testing domains. This chapter touches on both areas by creating a security infrastructure for the testing of WSN nodes, by:

1. Defining the layers of a WSN with the added testing capability of the testing protocol in Chapter 5.

2. Exploring the trade-offs of implementing various security policies against secure testing requirements, at every layer of the system.

3. Proposing a security scheme based upon design decisions that meet all applicable requirements.

4. Evaluating the proposed scheme against an expanded threat model, augmented with relevant physical and protocol considerations.

5. Comparing experimental results against WSN node-specific and network-level requirements, as well as other research.

Security is defined in Section 2.2 as both a network-level and node-specific requirement. In reality, secure design involves considering its principles on many additional sub-levels

of this. It has been suggested that the security of an embedded system can be seen as a pyramid of the following top-down abstraction layers [52]: protocol, algorithm, architecture, microarchitecture, and circuit. To incorporate wireless networking concerns into the mix, the WSN node testing being considered here will be split into the following amended set of top-down abstraction layers:

1. Testing protocol
2. Cryptographic protocol
3. Key management
4. Network topology
5. Architecture and microarchitecture
6. Physical (equivalent to circuit-level in [52])

The following discussion is separated into the aforementioned layers, where potential design choices and the implications of their implementation on a WSN are presented. Design choices are evaluated for suitability against the network-level and node-specific requirements of Section 2.2 as well as the test interface security requirements of Section 2.5.

4.1 Testing Protocol

The testing protocol of Chapter 5 outlines a procedure to be followed by both node and basestation in performing node tests using SBST programs. Before the protocol is initiated, the basestation key management scheme authenticates nodes and starts a session to exchange data. The testing protocol is only invoked under the context of this session. Upon receiving a *start_test* message, a node ensures that the message has arrived from its affiliated basestation, aborting the test otherwise. Buffer overflows of flash memory are averted by only allowing access to memory locations marked as available, and aborting the test if writing is attempted elsewhere. Even though all communications are encrypted and authenticated, an additional measure of data obfuscation is provided by the compression scheme used to reduce the size of SBST programs. The node stops communication for the duration of SBST testing and generates a test response signaling a pass or fail. This

test response is encrypted, authenticated, and sent to the basestation. If another test is required at a later time, and the node has not overwritten the SBST program in flash memory, subsequent *start_test* messages can be sent without the need to transfer more SBST programs. This averts unnecessary communication both from a security and energy preservation perspective. Many security considerations at the testing protocol level must therefore be taken into account when providing overall node security.

4.2 Cryptographic Protocol

In Section 2.4.1 both public-private key and symmetric key algorithms are introduced as cryptographic algorithms used to secure communications. A *cryptographic protocol* is a scheme for encrypting information that includes *cryptographic primitives* such as hashing, public-private, and symmetric algorithms as its constituent elements, and may define different modes of operation. The security of protocols themselves can be evaluated by employing known methods of code-breaking, or *cryptanalysis*, which attempt to find weaknesses in their internal components, such as the mathematical problems they are based upon. If a protocol is found to be secure, a *brute-force attack* can still attempt to find the correct key that produces a given plaintext by trying every possible key combination. Protocols that are *computationally secure* cannot be compromised with brute-force given the computing power available today or in the near future. For cryptographic algorithms without major security flaws, a comparison of the amount of "security" they provide can be performed. Table 4.1 shows an analysis comparing the popular symmetric key algorithm AES with the public-private key algorithm RSA, from information in [53].

Table 4.1: Equating the security of different key-sizes for the algorithms AES and RSA

Equivalent security		
AES 128 bit	AES 192 bit	AES 256 bit
RSA 3072 bit	RSA 7680 bit	RSA 15360 bit

It can be seen that the nature of public-private key algorithms requires a much greater key size to provide the same security as a symmetric-key algorithm. Since the requirements of WSN nodes necessitate applications of low computational intensity, public-key algorithms are rarely considered as viable on these platforms [54]. Recent work has nevertheless shown

that they are viable when implemented in hardware offering low power consumption [55], and in software while incurring a sizable energy cost [56]. The remainder of this section considers the use of symmetric key cryptographic protocols for use with securing the testing protocol of Chapter 5.

Network packets transferred over a WSN are usually small and their traffic is "bursty" in nature. This makes them well-suited for use with block ciphers, which are less computationally intense than stream ciphers. Fortunately, block ciphers are prevalent in practice, so a comparison of the characteristics of several candidates can be made based upon the test interface security requirements of Section 2.5.

The use of block ciphers on WSN platforms has been studied in the past, most notably the comparison of [57] on the algorithms RC5, RC6, Rijndael (also known as AES), MISTY1, KASUMI, and Camellia. Several comparisons are made between software implementation code size, energy, speed, and security. For the purposes of satisfying the security and node requirements, it is essential that the chosen block cipher have a minimal code size, minimal energy requirements, and maximum security. A comparison in [57] between the aforementioned block ciphers shows that the AES algorithm is the most efficient in terms of energy per byte of encrypted output. This result is consistent across both speed-optimized and size-optimized versions of the algorithms executed on a Texas Instruments MSP430F149 MCU, for all packet sizes typical of WSN node traffic. The security of AES is also found to be superior to other algorithms both by NIST, in their selection of Rijndael as the new AES (as explained in Section 1.2), and by the authors of [57]. Since then, extensive cryptanalysis has also not exposed significant vulnerabilities in the algorithm. However, the code size for AES is the largest of all algorithms compared. To mitigate this problem, the execution of software AES can be avoided by offloading the processing to a hardware AES core. In fact, one of the criteria for choosing Rijndael as AES was that small and efficient hardware implementations are possible. Hardware offloading reduces MCU power consumption, does not require any cryptographic MCU code, and hardware AES can presumably run faster than an MCU software version. Fortunately, hardware AES cores are available on-board WSN transceiver chips such as the Chipcon CC2420 used in the test setup of the experiment described in Section 4.8.

The standard AES algorithm can be used with 128-bit, 192-bit, and 256-bit key sizes. In [53], it is stated that the 128-bit key size can be used in applications to secure government communication until the year 2030 and beyond. A simple calculation of the amount of time

a brute-force attack would require to find a single correct key can be seen in Table 4.2.

Table 4.2: Time to complete a brute-force attack on a 128-bit AES key

Algorithm	Possible Keys	Optimized AES code [58]	Execution speed on P4-3.8 GHz (1 core)	Time to find key with 10^8 CPUs
AES 128-bit	2^{128}	237 cycles/block	$1.6 \cdot 10^7$ blocks/sec.	$6.7 \cdot 10^{15}$ years

The result shows that using the computing power of 100 million fast CPUs, it would take many times the age of the universe to find the correct key using a brute-force attack. Using greater key sizes would offer greater security, however, this is not required for the intended application since power consumption would suffer as a result. The 128-bit AES protocol is therefore used within the security infrastructure of this chapter. There are several modes of operation for block ciphers, and AES specifically, that offer different security advantages. Some common modes include:

1. Electronic Codebook (ECB) – Inputs to the encryption algorithm are a block of plaintext and a key, which produce a block of ciphertext.
2. Cipher Block Chaining (CBC) – Same as ECB mode, except the plaintext is XORed with the ciphertext of the previous encryption.
3. Counter (CTR) – A *nonce*, or counter-value is encrypted with a key, whose result is XORed with the plaintext.
4. Cipher Block Chaining Message Authentication Code (CBC-MAC) – Uses a block cipher to generate a hash of a message. Similar to CBC mode for encryption.
5. Counter with CBC-MAC (CCM) – Mode which combines both CTR and CBC-MAC modes such that each set of blocks is authenticated with a hash value.

In the ECB mode of operation each encrypted block is independent of the last and next blocks. If the same key is being used to encrypt successive blocks, the problem of *replay attacks* arises. Even though an adversary is not able to decrypt the data being communicated, he is able to fool the receiver by recording and replaying past blocks. The CBC mode offers a simple solution to this problem, past and present blocks are "chained" together by performing a logical XOR operation between the ciphertext of the last block

and the plaintext of the current block. If the block is the first in a series, an *initialization vector* (IV) is used instead. The receiver will attempt to recover the plaintext by performing an XOR with the ciphertext of block he last received. If he was prevented from receiving a block, or a rogue block was injected into the channel, the decryption will not be successful. The CTR mode is similar to CBC, except that a secret *counter value* is encrypted with the key, and the result placed into a logical XOR with the ciphertext. It is potentially simpler since each encryption can be performed independently of the last, while still retaining the "chaining" effect through use of the counter. Unlike the modes thus far described, the CBC-MAC mode generates a hash of several blocks in order to provide data integrity and authentication. It combines the idea of digital signatures in public-private key cryptography with that of a hashing algorithm, all implemented with the same block cipher used for encryption. It operates in much the same way as CBC mode, where the IV is set to zero and the output of each encryption is fed into the input of the next. However, a key specific to authentication is used. This ensures that confidentiality and authentication are separate and the compromise of one does not affect the other. The CCM mode combines CTR and CBC-MAC to provide both confidentiality without the possibility of message replay, as well as data integrity and authentication through the hashing each block set.

In choosing a mode for use in the security infrastructure, it should provide confidentiality, data integrity, and authentication, as per the requirements of Section 2.4. The CCM mode is therefore chosen to meet all requirements. This mode has also has hardware support on the CC2420 RF transceiver chip on-board the test setup.

4.3 Key Management

The discussion in Section 2.4.4 defines key management as the generation, establishment, distribution, and revocation of cryptographic keys. This section addresses each of these facets of key management under the requirements stated earlier.

Depending on the WSN type, the responsibilities of key management can be relegated to either the nodes themselves, a basestation, or any combination of the two. By their nature, ad-hoc networks must perform all activities on-board the node, may have policies for exchanging keys with neighbouring nodes, and some may expel nodes under suspicion of being attackers based upon network voting. Mesh networks have defined routing and can even be preloaded with some information to ease their responsibilities. Star networks

can relegate all key management duties to their basestation, a trusted entity, since it may be advantageous to achieving their requirements. For the network topology used in this security infrastructure, as defined in Section 4.4, basestations are left to manage keys which allows nodes to save energy, memory, and the implementation of cryptographic primitives in software. The following is a scenario for WSN key management which is mirrored in Figure 4.1, where hollow arrowsheads symbolize insecure links and solid arrowheads secure links:

(1) If there are n basestations and p nodes in the network, each node is pre-loaded with a unique *master key* and each basestation is preloaded with p/n master keys.

(2) The WSN is deployed and activated. Nodes establish communication with the basestation in their vicinity, transmitting their identifications without encryption.

(3) If a basestation does not have the master key for a node in its vicinity, it requests it from other basestations over the secure basestation network.

(4) Basestations swap keys until they all contain the keys required to communicate with the nodes in their vicinity.

(5) (a) Basestations generate p/n link keys, one for each node in their vicinity.

(b) Link keys are sent to each node to use with the current *session* and stored in volatile memory on-board the node.

(6) Communication takes place between nodes and the basestation within the context of the session, using link keys.

(7) Once M packets have been transferred between the basestation and a node, the basestation generates a new link key and distributes it to the node again using the corresponding master key.

(8) If new nodes are deployed, their master keys are pushed onto the basestation network by the deployer who has access. The basestations repeat steps 4 and 5b for only the new nodes.

(9) If any node should detect that it is being tampered with, it first broadcasts a message to the basestation announcing the tampering, and then erases its volatile memory

containing the master and link keys. The basestation removes the affected master and link keys permanently.

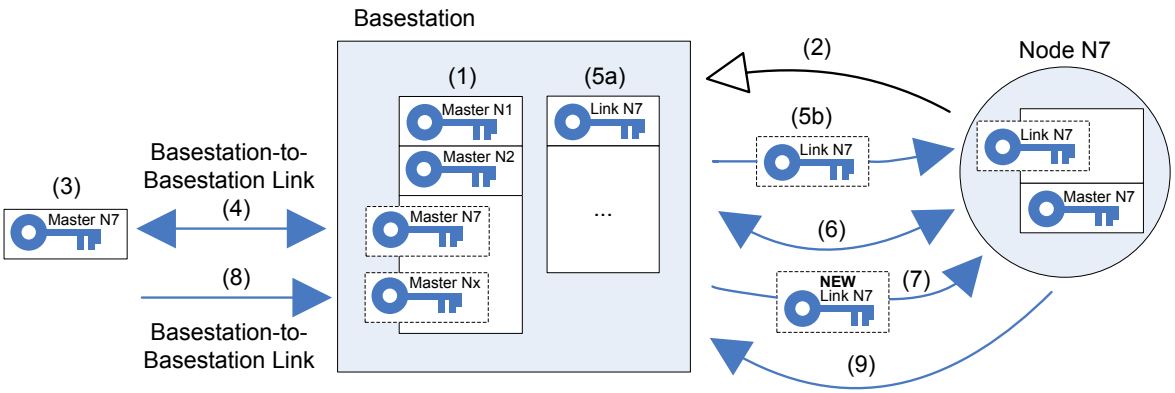


Fig. 4.1: Key Management Scenario

The basestation is considered a secure device that is more resistant to tampering than any node. This allows a basestation to retain a secret seed to use in a key generation schemes such as the SHA-256 hash algorithm. The seed can be based upon system time, bits in memory, some part of an activity log, or any number of sources which increase its entropy. Since only basestations contain a good pseudo-random number generator, key establishment can then be left to them, from where keys can be distributed to nodes. Schemes such as Diffie-Hellman key exchange are good at protecting against eavesdroppers, but are susceptible to the *man-in-the-middle attack*. Upon deployment of the network, an adversary can pretend to be a genuine node and establish a valid key with a basestation. Even though the star topology precludes the intruder from logging other network traffic, this is nevertheless not a desired outcome. The solution is to identify nodes to the basestation before deployment, and allow nodes to be later added through access to the secure basestation network. Since basestations do not have the same cost limits as nodes, they can be reasonably secured against physical tampering.

In step 1, the purpose of pre-loading basestations with only a subset of the master keys existing in the WSN is to save on their memory requirements. Any node will only need to communicate with a single basestation at any given time, so keys can be distributed and swapped as needed. Establishing link keys in step 5b provides for a changing key that is resistant to ciphertext-only cryptanalysis since it is used for only a limited amount of time,

or rather, a limited number of packets encoded with that single key. Even if vulnerabilities in the underlying cryptographic protocol are discovered in the future, the likelihood of uncovering the key by analyzing a small amount of traffic is diminished.

4.4 Network Topology

This section is concerned with determining a suitable network topology for securing WSN node testing data. Some common WSN topologies were presented in Section 1.1.2, which have advantages and disadvantages depending on their intended application.

An ad-hoc topology is useful in cases where the network structure is continually changing, or when deployment does not allow a pre-planned network structure, such as when nodes are dropped from an airplane over a wide area. Networks are designed to self-organize and self-assemble based upon some rules of behaviour. Since an extensive set of these rules must be defined, this topology presumably requires more node resources, such as memory, than methods with fixed topologies. Furthermore, ad-hoc security protocols are equally as complex and by extension, computationally intensive. Nodes are generally susceptible to impostor attacks at the moment the network is brought online, since there is no central authority in this topology.

Mesh networks differ from ad-hoc in that nodes rarely move after deployment and a fixed topology of the network can be presumed. They have many characteristics that are useful in WSN applications, by virtue of their interconnection to all other nodes within their vicinity. They offer maximum resiliency to node failures, since they are able to route packets to their other neighbours. Basestation nodes are also unnecessary, saving cost in total network deployment cost. However, asymmetrical power consumption is an issue where some nodes on the outer edges in the network consume less energy than nodes which actively route other nodes' packets. If nodes are unaware of their neighbours, this topology is also susceptible to the same impostor attacks as ad-hoc networks. Some key distribution schemes preload each node with link-keys for every other node, for which node memory requirements do not scale well with network size. Other key distribution schemes preload only a subset of link-keys, which scales slightly better but whose resiliency to failure suffers since not every node can communicate with any other.

Star networks eliminate the need for establishing trust in other network nodes. All traffic is sent to a centralized basestation, which is the trusted authority. The basestation

communicates with all nodes directly, and can be preloaded with link-keys to each node. Since basestations can have more resources such as processing capacity, memory, and batteries than nodes, and can be made more physically secure than nodes, they are well-suited to the task. Network resiliency suffers if a basestation fails, but basestations can be built with more reliable components. Impostor attacks, nodes for which the basestation does not have preloaded keys, can be ignored.

The topology suggested for use here, to satisfy all aforementioned requirements, is a hybrid of a star and mesh network, such as seen in [59]. It can be imagined as many star networks connected by a mesh between the basestations, as seen in Figure 4.2. A network consists of many nodes and few basestations, which can be deployed in any fashion. Upon bringing the network online, nodes can use their *received signal-strength indicator* (RSSI) to determine the basestation closest to them. They would then set up link-keys with that basestation for future communications based upon the procedure in Section 4.3. Basestation are able to relay sensor readings within the *basestation network*, consisting of mesh links between all basestations. This network is secured with similar preloaded keys between all basestations. Basestations would be designed for reliability, but if one were to fail, traffic on the basestation network could be redirected around the failure. This hybrid topology offers the advantages of both the star and mesh topologies.

4.5 Architecture and Microarchitecture

The architectural and microarchitectural abstraction layers of security refer to its structuring of software and hardware, respectively. Secure design principles dictate that hardware-software partitioning should be motivated by security rather than other factors. Instead of securing the entire system, only sensitive IP should be placed in a secured portion of both hardware and software. Portions of the system not containing secrets can be secured by making a dependency on a portion of secure hardware, such as the *trusted platform module* in *trusted computing*.

Nodes suffer from an obvious dilemma in securing their architectural design: they are often composed of discrete COTS chips, whose designs are often unknown and cannot be modified to ensure security is maintained. Even if secure chips are employed, they are often assembled on a common low-cost PCB, whose traces can be probed. For example, the WSN research platform used in Section 4.8 uses a low-power MCU chip and an RF

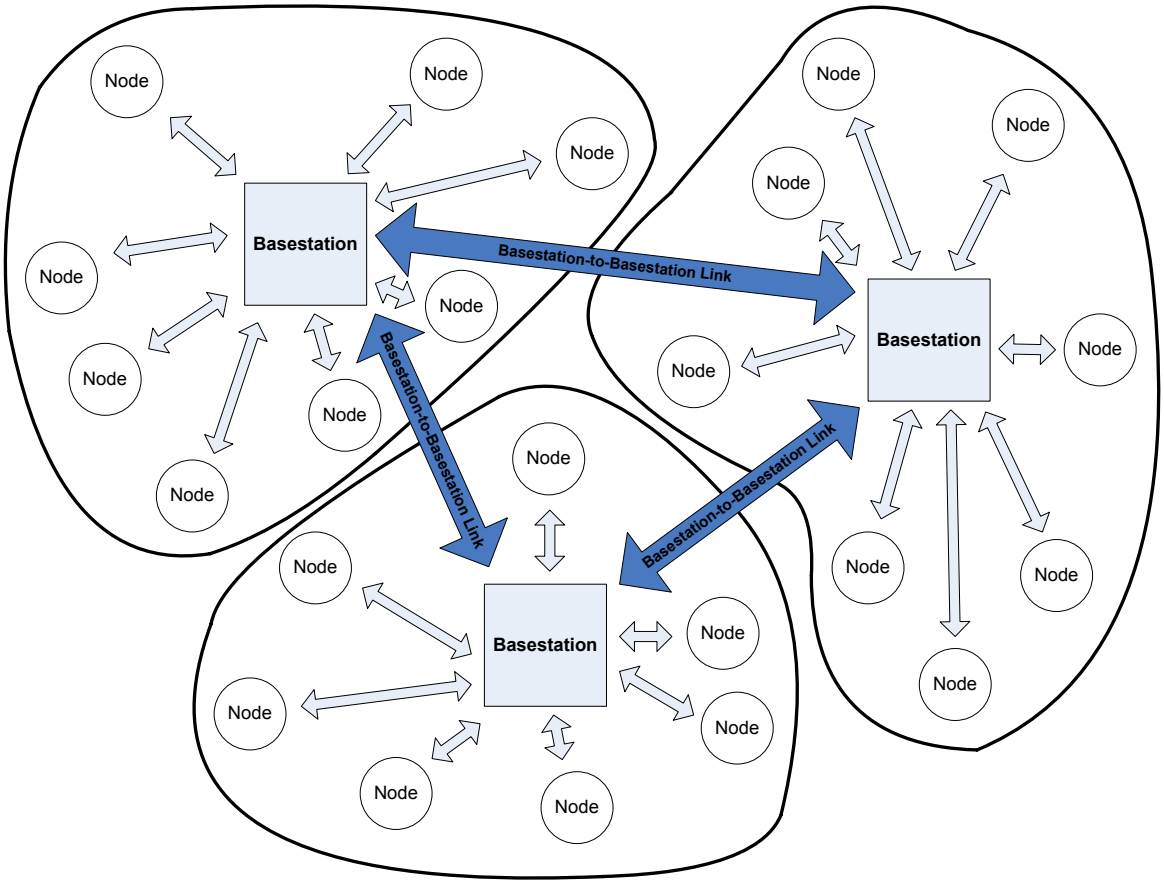


Fig. 4.2: Hybrid Network Topology

transceiver chip with cryptographic capabilities. The RF transceiver chip is not designed by the manufacturer with secure principles in mind. It is possible for anyone employing the correct protocol to probe the memory of the chip where the secret keys reside. However, the key revocation scheme mentioned in Section 4.3 relies on sensors at the microarchitectural level to detect physical node disturbances. A disturbance would bypass the MCU and directly cut power to the RF transceiver for a short period, deleting the secret key in the process.

Data communication between the transceiver and MCU is likewise insecure since all cryptographic processing between the node and network is performed within the transceiver. A more favorable approach to WSN node design would combine the MCU and cryptographic chip within the same system-on-chip (SoC) package or die, seen in Figure 4.3. This portion of the system would be secure, while the transceiver could reside elsewhere on the same

board. If such an SoC is not possible under the various node requirements and constraints, cryptographic functions could be implemented in software on the MCU to prevent unencrypted data buses from leaving the secure area of the chip. However, such a solution would certainly require an MCU with more flash and RAM, at increased cost.

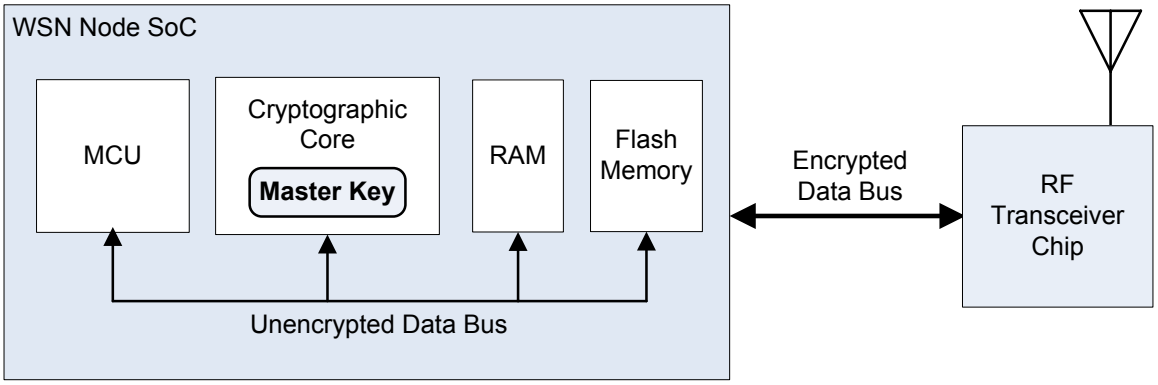


Fig. 4.3: Secure WSN Node Architecture

To secure system software, it can be placed in a non-volatile medium such as ROM to ensure it cannot be modified by an attacker. This would require thorough testing before deployment, since software updates or upgrades would be impossible. The nature of the testing scheme proposed in Chapter 5 makes this method unsuitable for use here, as SBST programs are directly executed from on-board flash memory. If possible, flash memory can instead be located on-board a secure SoC as shown in Figure 4.3. If needed it can also reside outside of the secured portion of the system, and a hashing algorithm can be used to check its fingerprint against that of an unmodified version of the software.

4.6 Physical Security

The physical vulnerabilities of a WSN node are limited to the secure portion of the system, represented by the SoC in Figure 4.3. Physical attacks can be launched based upon many chip characteristics, such as power consumption, electromagnetic emissions, and acoustic emissions over time. Power consumption attacks are most common, and both *simple power analysis* and *differential power analysis* have proven effective at uncovering secret keys in cryptographic chips. This proves more difficult on complex SoCs, but is nevertheless possible. Countermeasures against such attacks include equalization of chip power consumption

over time. This can be done by utilizing a transistor family such as *masked dual pre-charge logic* (MDPL) in lieu of CMOS, but at a heavy cost in area, speed, and power.

Since the COTS chips used in many WSN nodes are not created with security goals in mind, and since the manufacture of a custom SoC is very costly, physical node security is an area that cannot be addressed without compromising the node-level requirements of Section 2.2.

4.7 Proposed Security Scheme Summary

Based upon the preceding discussion of this chapter, the proposed security scheme to meet security, network, and node requirements is summarized here. The following is a list of system layers, with relevant security decisions and considerations:

1. Testing protocol – confidentiality and authentication defined within the context of a session, buffer overflows avoided, unnecessary communication averted.

2. Cryptographic protocol – AES block cipher offers resiliency to cryptanalysis, infeasibility of brute-force attack, energy efficiency at the protocol level, and extensive modes to provide both confidentiality and authentication.

3. Key management – proposed scheme places responsibility for key management on basestations which can be built more securely than nodes, scheme is not susceptible to man-in-the-middle attack, allows new nodes to be deployed, and makes provisions for revoking keys of nodes that are tampered with.

4. Network topology – nodes in hybrid star/mesh network place trust in only one entity, while the network retains scalability and resiliency against failure.

5. Architecture and microarchitecture – contrary to secure design principles, RF transceiver and MCU are connected by an unsecured bus to save energy on offloading cryptographic functions, but tamper detection sensors delete secret keys in case of attack.

6. Physical (equivalent to circuit-level in [52]) – impossible to design custom chips employing power consumption equalization under node constraints, these methods are left to the most security sensitive devices.

It is apparent that the proposed security scheme considers the attacks proposed by the Dolev-Yao threat model, as well as others. Even though not all potential attacks can be countered under the cost and power constraints of WSN nodes, the proposed security scheme aims to offer strong resilience to attack within these constraints. Determining the precise security strength of the proposed scheme requires a deeper cryptanalysis, which is a sizable undertaking that has been left to future work. Rather, focus is shifted in Section 4.8 to verifying that the scheme meets node requirements in two experiments measuring power consumption.

4.8 Security Scheme Energy Expenditure Results

In Section 2.2, it is said that nodes should have low power consumption so that they should be able to last months or years on a single battery. The requirement that nodes be secured against attack is also put forth as a priority, as is the need for periodic testing in Section 2.2.1. Since a robust security scheme is proposed in Section 4.7 for use on the testing protocol of Chapter 5, it should be ensured that it does not consume an excessive amount of energy in its implementation. This experiment has two goals:

1. Quantifying the energy required to perform cryptographic functions on the CC2420 RF transceiver.

2. Determining the energy overhead of implementing the proposed security scheme on top of the testing protocol.

The first part of the experiment involves finding the energy required for a single block of encryption, which is found by instructing the node to perform encryption while placing the MCU in a low-power sleep state. The same experimental setup from Section 3.4.1 is used, although here the AES encryption capabilities of the Chipcon CC2420 RF transceiver chip are harnessed. The current measurement circuit also described in Section 3.4.1 is used in the same way to find the energy consumption of performing encryption on a single block of data. Results can be seen in Table 4.3, where they are compared to other implementations in the literature on the basis of energy per bit of encryption. The second part of the experiment extends the first experiment by finding the amount of energy required to perform AES cryptography on testing data. The portion of energy consumption used in cryptography is called the overhead of the security scheme, the results of which can be seen in Table 4.4.

It should be noted that the CCM mode of AES decryption is not functional on the engineering sample of the Chipcon CC2420 RF transceiver chip that is part of the WSN node research platform. The chip instead offers a working version of the ECB encryption mode, which is the most common of all modes and allows for the energy comparison of Table 4.3. The energy needed for CCM encryption is estimated by multiplying the number of ECB encryptions by a constant based upon the details of CCM mode operation. For every block of data that is to be encrypted, a CTR as well as a CBC-MAC operation must be executed. However, the result of every 8 blocks of CBC-MAC must itself be encrypted and sent. From that, the amount of energy required for a typical communications scenario in CCM mode can be determined to be 2.125 times that of the same in ECB mode. It is also noteworthy that encryption is considered here instead of the decryption of SBST data that takes place on the node. Since a hardware implementation of the AES algorithm is very symmetrical in performing both encryption and decryption, here it is assumed that both require the same amount of energy. Similar assumptions have been observed in the literature for energy calculations [60].

Table 4.3: Energy Comparison of AES 128-bit ECB Encryption

Platform for AES 128-bit Encryption (ECB Mode)		Energy/bit (nJ)	Investment Cost
Software Implementations	Mica2dot WSN research platform [56]	202.5	Low
	iPAQ H3670 Pocket PC [61]	151.25	Low
	PPT2800 Pocket PC [60]	66.6	Low
Hardware Implementations	Coprocessor on FPGA [62]	80.469	Low
	CC2420 RF transceiver (*this thesis*)	**2.425**	**Low**
	Custom ASIC in 0.13μm CMOS [63]	0.080	High

4.9 Discussion

The first portion of the experiment determines the energy efficiency of performing AES cryptography on the WSN research platform, and compares it to other platforms from the literature. From the results it can be seen that the ability to offload AES functions to the research platform transceiver chip proves advantageous in terms of both energy

Table 4.4: Overhead of Security Scheme on Wireless Transfer of Tests/Responses

Function		Energy Usage (mJ)		
		P_1	P_2	P_3
Security Scheme	Decryption of SBST Program (Equivalent to CCM Mode)	0.281	0.015	0.181
	Encryption of Test Response (Equivalent to CCM Mode)	$6.596 \cdot 10^{-4}$	$6.596 \cdot 10^{-4}$	$6.596 \cdot 10^{-4}$
Testing Scheme	Wireless Reception of Compressed SBST Program	11.955	0.603	7.167
	Wireless Transmission of Test Response	0.074	0.074	0.074
	Overhead of Security Scheme on Testing Scheme	**2.34%**	**2.31%**	**2.51%**

consumption and initial investment cost. Only the custom ASIC gives a smaller power consumption than CC2420 cryptography, although the initial investment cost of producing such a chip would be very high. While it is quite likely that there are COTS chips with AES cores that are low-cost and use slightly less power, the addition of another discrete chip to the WSN node would increase the production cost of the PCB and increase the physical footprint of the device. Thus, the device best balancing the cost and energy needs of a node would be the CC2420-based solution. However, if physical security is of utmost importance, cryptography should be implemented in software on-board the MCU as per architectural secure design principles. The added cost of an MCU containing more flash and RAM would enable such an implementation, and would pose more resistance to a physical attack, since unencrypted data would not be transmitted onto the bus between the MCU and RF transceiver. The added power consumption of such a scenario is not deemed cost-effective for the purposes of this chapter, and the design decisions are made in favor of preserving energy.

The second portion of the experiment shows the overhead of adding security, in this case the encryption and decryption of testing data, to the testing scheme of Chapter 5. From the results it is confirmed that the transmission and reception of data poses the greatest energy consumption to the system. When cryptographic functions are offloaded to the transceiver

chip, the encrypting and decrypting of a network packet presents about a 2.5% overhead on the energy consumption of transmitting or receiving that packet. This confirms that a moderately strong security scheme can be implemented without excessive overhead of energy consumption on WSN nodes. Of course, had the potentially more secure software-based cryptography scheme been implemented, the overhead would have most certainly been higher.

This chapter outlines a security infrastructure for use with the WSN node testing protocol of Chapter 5. It defines security in the relevant layers of a WSN, exploring the possibilities and trade-offs of implementing various security measures. The selections that are made are a balance between the security, network, and node requirements of WSNs. Each layer of the system has been evaluated against the relevant threats present at that level, under the direction of the Dolev-Yao threat model as well as that of the physical-level attacker.

Chapter 5

Combining Testing, Compression, and Security

The discussion of Section 2.2.1 establishes the need for periodic node testing to ensure an acceptable WSN availability. In Section 2.3 the idea of using SBST programs to perform this testing is introduced as having many advantages in the case of WSN nodes. This chapter outlines a protocol that allows for testing to take place. If defines the nature of the communication between basestation and node that leads to successful testing, which achieves the network availability requirement. This high-level protocol is compatible with the methods of SBST program compression and test security introduced in Chapters 3 and 4, respectively.

5.1 Testing Protocol

The purpose of the testing protocol described here is to dynamically distribute and execute SBST tests on WSN nodes, and to collect the results of those tests. It is said in Section 4.4 that the hybrid star-mesh network topology has advantages in providing testing security over other topologies. The basestations in this network model can also act as SBST repositories and coordinate the testing process. Since a typical basestation has greater memory resources than a node, it can store an array of SBST tests in its memory and distribute tests as required. This allows the small flash memories of WSN nodes to store individual tests as required, and does not necessitate each node simultaneously storing all SBST tests it would eventually execute.

A visual representation of the testing protocol applied to the testing of a single node can be seen in Figure 5.1, which corresponds to the following steps:

(1) As the WSN comes online, nodes join the basestation in their vicinity and set up a secure communication session for data exchange (as described in Section 4.3).

(2) (a) Each basestation is deployed with a repository of SBST programs used for the testing of various node systems.

(b) Should there be a need to add to this repository, the deployer who has access to the basestation network can multicast new tests to the basestations.

(3) Based upon predefined rules for periodic testing, a basestation transmits an SBST test (denoted P_n) to a node. The SBST programs are compressed by the BSTW algorithm and Rice coding. Data confidentiality and authentication is ensured through the use of the AES encryption algorithm in CCM mode, as well as other elements of secure design described in Chapter 4.

(4) As the node is receiving the SBST program, the data is decrypted and authentication verified. Once a decrypted packet is in the buffer, it is decompressed by first applying inverse Golomb-Rice coding, then the BSTW algorithm for data reordering. On-the-fly, the decompressed SBST program is written to an area of flash memory labeled as available.

(5) Once the SBST program has been distributed, the node waits for a *start_test* message from the basestation. Once received, the node stops communication with the basestation and executes the test.

(6) The result of each test of the SBST program is compared with a result known to be correct also stored within the program, recording each failure. A single *test_result* message is formulated with a list of the tests that failed, and the identification number of the SBST program.

(7) The *test_result* is encrypted on the node and sent back to the basestation. The basestation records the result for future reference by the network deployer. Reviewed regularly, such records allow failing or failed nodes to be replaced before they impact network availability.

(8) (a) If the basestation should require a node to rerun the same test at a later time, the *start_test* message can be sent without retransmitting the SBST program.

(b) If different tests are to be executed, the node writes the additional SBST programs into its flash memory until memory is full. Older tests are overwritten based upon a least-recently-used (LRU) replacement scheme.

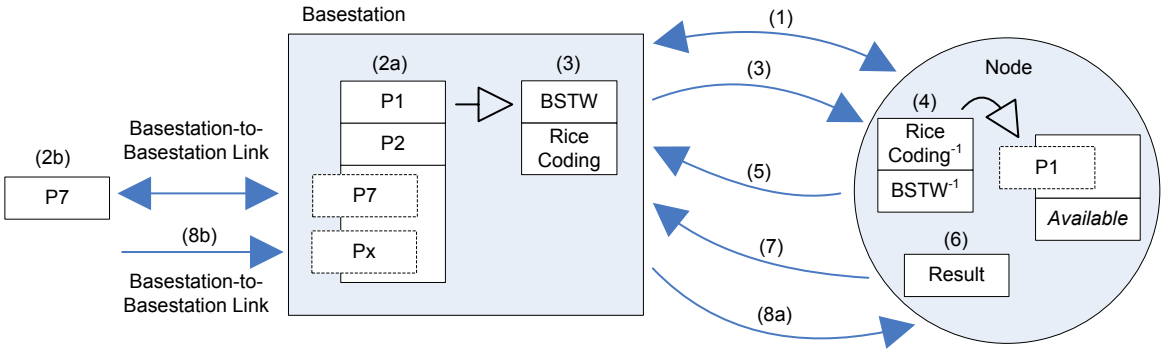


Fig. 5.1: WSN Node Testing Protocol

An advantage of this dynamic model of test distribution is that it allows for SBST tests to be updated and added to the basestation repository after deployment has taken place. In deploying SBST programs, nodes themselves are responsible for storing the programs and keeping track of available flash memory locations. The test commands sent by the basestation are high-level, so the tracking of all node memory spaces by the basestation is not required. This allows the testing of a WSN with nodes of heterogeneous configurations, memory resources, and sensor capabilities using the same testing protocol. Of course, the basestation must associate SBST programs with their corresponding node configurations.

5.2 SBST Tests Types

The testing protocol presented in Section 5.1 focuses on the self-testing of a node by a basestation. As seen in Figure 5.2a, such tests are applied by a *node-under-test* (NUT) MCU to the system buses, memory, sensors, peripherals, RF transceiver, and MCU itself. They are built to test an entire node, and for each group of testable components an SBST program can be created which offers acceptable fault coverage. For example, embedded RAM and flash memory are tested by March-family algorithms [64] implemented with MCU

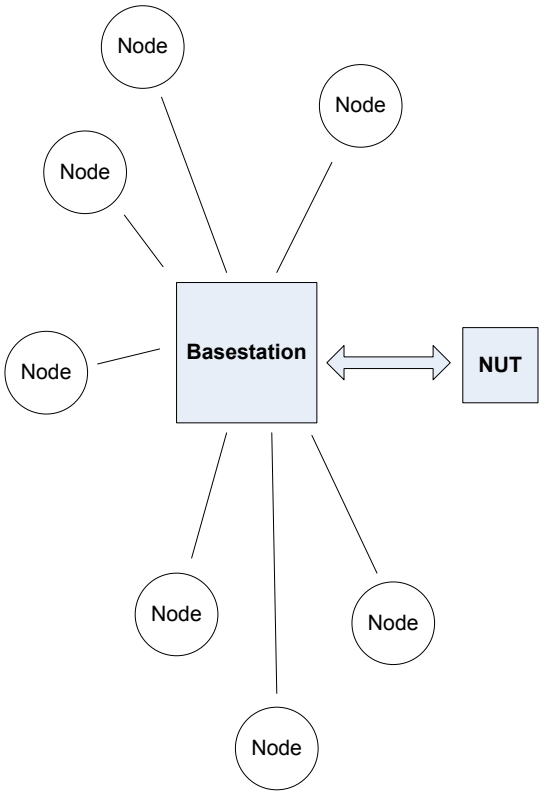
instructions. Self-tests are structured such that they minimize the possibility for masking MCU faults when comparing results known to be correct with actual results. These MCU faults can manifest themselves in various ways, the most severe of which prevents the formulation of a correct test response. In this case, the basestation can interpret a garbled result as a failed node and choose to periodically rerun tests to uncover intermittent faults.

In a finer-grained scenario of testing a node, the performance of the RF transceiver and antenna assembly can be evaluated by tracking their Tx and Rx gains over time [19]. This involves using the basestation to distribute different SBST tests to a NUT and several other nodes (denoted N_1 and N_2) affiliated with the basestation, as seen in Figure 5.2b. The NUT is first instructed to broadcast data, and the neighbouring nodes to receive it. The content of the data is discarded, but nodes N_1 and N_2 record RSSI and determine the gain of the transmitter as measured at the receiver. The process is then reversed, as nodes N_1 and N_2 are instructed to transmit and the NUT to receive. Again, the transmitted gain is recorded at the receiver. This testing scenario can predict nodes with failing batteries, RF transceivers, and antennas, if performance is tracked over a period of time. However, for simplicity and consistency, the *test_result* sent back to the basestation can include a comparison to a gain that is deemed acceptable based upon a threshold. Further details on the composition of SBST programs for WSN nodes can be found in [15, 19].

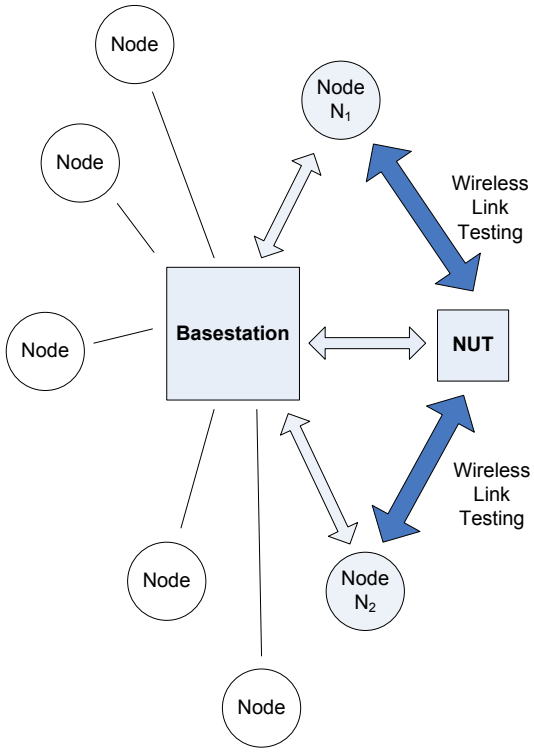
5.3 Discussion

This chapter presents a protocol to manage the testing of WSN nodes using SBST programs. The protocol works at a high level, making its use suitable for heterogeneous WSNs. It includes provisions for the SBST program compression introduced in Chapter 3 and the test security infrastructure seen in Chapter 4.

Improvements left to future work include the multicasting of SBST tests from basestation to nodes. This would decrease basestation energy requirements but would require an addition to the security infrastructure to manage *group keys* among nodes. The addition of a more complex node-performance tracking system on basestations would allow a finer degree of node failure prediction or even fault diagnosis. Measuring RF assembly performance, battery life, and intermittent fault frequency would also require greater memory on-board basestations, and may not prove cost-effective. However, these failure prediction methods could warrant further study.



(a) Node Self-testing with SBST Programs



(b) NUT RF Assembly Testing with Nodes N_1 and N_2

Fig. 5.2: Two Scenarios for SBST Testing: (a) Node Self-testing with SBST Programs, and (b) NUT RF Assembly Testing with Nodes N_1 and N_2

Chapter 6

Conclusion

This thesis presents a platform for the testing of wireless sensor network nodes. Chapter 2 describes the composition of a node, its requirements, and outlines the need for in-field periodic node testing to maintain network availability. A review of methods of system-level testing suggests software-based self-test programs are efficient for node testing. Remote testing raises the question of security against attackers, which leads to an introduction to communications security and modern methods of securing a testing interface. Also examined is the trend toward reducing test data volume between a tester and device-under-test, which prompts a general introduction into compression theory. Chapter 3 characterizes software-based self-test programs and evaluates several general-purpose compression algorithms and coding techniques for suitability in compressing them. A compression technique chosen based upon its low memory requirements is then used to quantify the energy savings yielded in transferring compressed test programs over uncompressed ones. Chapter 4 uses concepts of secure system design to protect the remote testing interface from attack. Design choices are explored at every layer of the system, and choices made based upon node and security requirements. The chosen security platform compares favorably against similar platforms for energy usage, and is shown to add only approximately 2.5% to the energy requirements of performing node testing without security. Chapter 5 defines the workings of the protocol which tests nodes for failure and reports back results to a basestation. Different test types are defined, which in turn use the compression and security platforms to reduce energy consumption and protect the testing interface against attackers, respectively. A full wireless sensor network node testing solution is thus presented that increases network availability by detecting and predicting node failures through periodic

testing. The solution also decreases node energy consumption by transferring compressed software-based self-test programs, and provides confidentiality and data integrity on the testing interface with minimal energy overhead.

References

[1] E. H. Callaway, *Wireless sensor networks: architectures and protocols.* Boca Raton, FL: Auerbach Publications, 2004.

[2] *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Std. 802.15.4, 2006.

[3] D. E. Sanger and J. Clausing, "U.S. Removes More Limits on Encryption Technology," *New York Times*, Jan. 13, 2000.

[4] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting mobile communications: the insecurity of 802.11," *Proceedings of the 7th annual international conference on Mobile computing and networking*, pp. 180–189, 2001.

[5] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)," Technical Report. FIPS-197, 2001.

[6] Electronic Frontier Foundation (EFF), *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*, 1st ed. Sebastopol, CA: O'Reilly and Associates, 1998.

[7] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Local and Metropolitan Area Networks*, IEEE Std. 802.11i, 2004.

[8] M. W. Chiang, Z. Zilic, K. Radecka, and J.-S. Chenard, "Architectures of increased availability wireless sensor network nodes," *International Test Conference (ITC), Proceedings of.*, pp. 1232–1241, 2004.

[9] P. K. Lala, *Self-checking and fault-tolerant digital design*, 1st ed. San Francisco, CA: Morgan Kaufmann, 2001.

[10] D. J. Smith, *Reliability engineering.* New York, NY: Barnes & Noble, 1972.

[11] S. Hariri and H. Huitlu, "Hierarchical modeling of availability in distributed systems," *Software Engineering, IEEE Transactions on.*, vol. 21, pp. 50–56, Jan. 1995.

[12] C. S. Raghavendra, V. K. P. Kumar, and S. Hariri, "Reliability analysis in distributed systems," *Computers, IEEE Transactions on.*, vol. 37, pp. 352–358, Mar. 1988.

[13] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli, "Fault tolerance in wireless sensor networks," in *Handbook of Sensor Networks*, M. Ilyas and I. Mahgoub, Eds. CRC Press, 2004, ch. VIII.

[14] A. Krstic, W.-C. Lai, K.-T. Cheng, L. Chen, and S. Dey, "Embedded software-based self-test for programmable core-based designs," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 18–27, July–Aug. 2002.

[15] R. Zhang, Z. Zilic, and K. Radecka, "Energy efficient software-based self-test for wireless sensor network nodes," *VLSI Test Symposium (VTS), Proceedings of.*, pp. 186–191, 2006.

[16] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, "Software-based self-testing of embedded processors," *Computers, IEEE Transactions on.*, vol. 54, pp. 461–475, Apr. 2005.

[17] A. Paschalis and D. Gizopoulos, "Effective software-based self-test strategies for online periodic testing of embedded processors," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on.*, vol. 24, pp. 88–99, Jan. 2005.

[18] L. Chen and S. Dey, "Software-based self-testing methodology for processor cores," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on.*, vol. 20, pp. 369–280, Mar. 2001.

[19] R. Zhang, "Energy reduced software-based self-testing for wireless sensor network nodes," M. Eng. thesis, McGill University, Montreal, Quebec, Canada, Oct. 2005.

[20] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inform. Theory*, vol. 29, no. 2, pp. 198–208, Mar. 1983.

[21] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[22] A. Odlyzko, "Discrete logarithms: The past and the future," *Designs, Codes and Cryptography*, vol. 19, no. 2-3, pp. 129–145, Mar. 2000.

[23] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.

[24] V. S. Miller, "Use of elliptic curves in cryptography," *Advances in Cryptology–CRYPTO '85 (Lecture Notes in Computer Sciences)*, vol. 218, pp. 417–426, 1986.

[25] W. Diffie and M. Hellman, "New directions in cryptography," *Information Theory, IEEE Transactions on.*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[26] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment," *Design Automation Conference, Proceedings of.*, pp. 151–155, 2001.

[27] B. Yang, K. Wu, and R. Karri, "Secure Scan: A Design-for-Test Architecture for Crypto Chips," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on.*, vol. 25, pp. 2287–2293, Oct. 2006.

[28] D. Hely, M.-L. Flottes, F. Bancel, B. Rouzeyre, N. Berard, and M. Renovell, "Scan Design and Secure Chip," *On-Line Testing Symposium (IOLTS), Proceedings of the International.*, pp. 219–224, 2004.

[29] D. Hely, F. Bancel, M.-L. Flottes, and B. Rouzeyre, "Secure Scan Techniques: A Comparison," *On-Line Testing Symposium (IOLTS), Proceedings of the International.*, pp. 119–124, 2006.

[30] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic, "Securing Scan Design Using Lock & Key Technique," *Defect and Fault Tolerance in VLSI Systems. Proceedings of the International Symposium on.*, pp. 51–62, 2005.

[31] Semiconductor Industry Association, "Test and test equipment, International Technology Roadmap for Semiconductors," 2005.

[32] A. Khoche, E. Volkerink, J. Rivoir, and S. Mitra, "Test Vector Compression Using EDA-ATE Synergies," *VLSI Test Symposium (VTS), Proceedings of.*, pp. 97–102, 2002.

[33] H.-G. Liang, S. Hellebrand, and H.-J. Wunderlicht, "Two-dimensional test data compression for scan-based deterministic BIST," *International Test Conference (ITC), Proceedings of.*, pp. 894–902, 2001.

[34] David Salomon, *Data Compression: The Complete Reference*, 3rd ed. New York, NY: Springer, 2004.

[35] D. J. C. MacKay, *Information theory, inference, and learning algorithms.* Cambridge, UK: Cambridge University Press, 2004.

[36] A. Moffat and A. Turpin, *Compression and Coding Algorithms.* Norwell, MA: Kluwer Academic Publishers, 2002.

[37] D. A. Lelewer and D. S. Hirschberg, "Data compression," *ACM Computing Surveys*, vol. 19, no. 3, pp. 261–296, 1987.

[38] Texas Instruments Corp. (2007, Feb.) MSP430 Ultra-Low Power Microcontrollers, MSP430x1xx – Flash ROM No LCD – Price list per 1000 units. [Online]. Available: http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?familyId=911&sectionId=95&tabId=1527&family=mcu

[39] T. A. Welch, "A technique for high-performance data compression," *IEEE Computer*, vol. 17, no. 6, pp. 8–19, June 1984.

[40] T. A. Welch, "High speed data compression and decompression apparatus and method," U.S. Patent 4 558 302, Dec. 10, 1985.

[41] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, Sept. 1952.

[42] N. Faller, "An Adaptive System for Data Compression," *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*, pp. 593–597, 1973.

[43] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Trans. Inform. Theory*, vol. 24, no. 6, pp. 668–674, Nov. 1978.

[44] D. E. Knuth, "Dynamic Huffman Coding," *Journal of Algorithms*, vol. 6, no. 2, pp. 163–180, June 1985.

[45] J. S. Vitter, "Design and analysis of dynamic Huffman codes," *Journal of the ACM*, vol. 34, no. 4, pp. 825–845, Oct. 1987.

[46] T. S. Han and K. Kobayashi, *Mathematics of Information and Coding*. Providence, RI: American Mathematical Society, 2001.

[47] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei, "A locally adaptive data compression scheme," *Communications of the ACM*, vol. 24, no. 4, pp. 320–330, Apr. 1986.

[48] S. W. Golomb, "Run-length encodings," *IEEE Trans. Inform. Theory*, vol. 12, no. 3, pp. 399–401, July 1966.

[49] R. F. Rice, "Some practical universal noiseless coding techniques," Jet Propulsion Laboratory, Pasedena, CA, Technical Report. Publication 79-22, Mar. 1979.

[50] S. Toub. (2002, July) Adaptive Huffman Compression. [Online]. Available: http://www.gotdotnet.com

[51] "CC2420 data sheet," revision 1.4, Chipcon AS, Oslo, Norway, 2004.

[52] D. D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, "Securing embedded systems," *IEEE Security and Privacy Magazine*, vol. 4, no. 2, pp. 40–49, Mar.–Apr. 2006.

[53] National Institute of Standards and Technology (NIST), "Recommendation for Key Management Part 1: General (Revised)," Special Publication 800-57, May 2006.

[54] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, Sept. 2002.

[55] G. Gaubatz, J.-P. Kaps, E. Ozturk, and B. Sunar, "State of the art in ultra-low power public key cryptography for wireless sensor networks," in *IEEE International Conference on Pervasive Computing and Communications Workshops*, 2005, pp. 146–150.

[56] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks," *Pervasive Computing and Communications, IEEE International Conference on.*, pp. 324–328, 2005.

[57] Y. W. Law, J. Doumen, and P. Hartel, "Benchmarking block ciphers for wireless sensor networks," *IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pp. 447–456, 2004.

[58] K. Aoki and H. Lipmaa, "Fast Implementations of AES Candidates," *Third AES Candidate Conference*, pp. 13–14, 2000.

[59] S.-P. Chan, R. Poovendran, and M.-T. Sun, "A key management scheme in distributed sensor networks using attack probabilities," *IEEE Global Telecommunications Conference*, pp. 1007–1011, 2005.

[60] R. Karri and P. Mishra, "Minimizing energy consumption of secure wireless session with QoS constraints," *Communications. IEEE International Conference on,*, pp. 2053–2057, 2002.

[61] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "Analyzing the Energy Consumption of Security Protocols," *Low Power Electronics and Design, Proceedings of the International Symposium on.*, pp. 30–35, 2003.

[62] A. Hodjat and I. Verbauwhede, "Interfacing a high speed crypto accelerator to an embedded CPU," *Signals, Systems and Computers, Conference Record of the Thirty-Eighth Asilomar Conference on.*, pp. 488–492, 2004.

[63] G. Bouesse, M. Renaudin, A. Witon, and F. Germain, "A clock-less low-voltage AES crypto-processor," *Solid-State Circuits Conference, Proceedings of the 31st European.*, pp. 403–406, 2005.

[64] A. van de Goor, "Using March Tests to Test SRAMs," *IEEE Design & Test of Computers*, vol. 10, no. 1, pp. 8–14, Mar. 1993.