1-1-2008

Security in wireless sensor networks : implementatin of Misty1 cipher in TinySec library

Ajay P. Mamtura
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>

Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Mamtura, Ajay P., "Security in wireless sensor networks : implementatin of Misty1 cipher in TinySec library" (2008). *Theses and dissertations*. Paper 298.

This Thesis Project is brought to you for free and open access by Digital Commons @ Ryerson. It has been accepted for inclusion in Theses and dissertations by an authorized administrator of Digital Commons @ Ryerson. For more information, please contact bcameron@ryerson.ca.

TK
7872
DL
M36
2008

**SECURITY IN WIRELESS SENSOR NETWORKS:
IMPLEMENTATION OF MISTY1 CIPHER IN TINYSEC LIBRARY**

by

Ajay P. Mamtura, Bachelor of Engineering (Electronics)
University of Mumbai (Bombay), India, 2002

A project
presented to Ryerson University

in partial fulfillment of the
requirements for the degree of
Master of Engineering
in the Program of
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2008

© Ajay P. Mamtura 2008

DECLARATION

I hereby declare that I am the sole author of this project.

I authorize Ryerson University to lend this project report to other institutions or individuals for the purpose of scholarly research.

Ajay Mamtura

I further authorize Ryerson University to reproduce this project report by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Ajay Mamtura

SECURITY IN WIRELESS SENSOR NETWORKS: IMPLEMENTATION OF MISTY1 CIPHER IN TINYSEC LIBRARY

by

Ajay P. Mamtura (amamtura@ryerson.ca , student ID 04 905 2392)

Master of Engineering, Electrical and Computer Engineering, Ryerson University, 2008

Supervisor: Dr. Truman Yang

ABSTRACT

Communication security is key to widespread adoption of wireless sensor networks. Coupling this need with the typical resource constraints encountered, TinySec was introduced by a group from UC Berkeley for the open source and highly popular TinyOS - a platform designed for wireless embedded sensor networks. Prior to this project the TinySec security implementation was provided with 2 ciphers, namely Skipjack and RC5. The library being modular in design afforded the opportunity to replace the cipher subcomponent with one of our choice, Misty1 in our case. Misty1 is a block cipher based on nested Feistel network with variable number of rounds (typically 8). It uses a 128-bit sized cryptovvariable and block size of 64 bits. Designed on the basis of the theory of provable security against differential and linear cryptanalysis, the cipher is intended to allow realization of high-speed encryption on hardware platforms as well as on software environments. The new cipher implementation within TinySec was tested using 2 sample applications and run inside TOSSIM – a discrete event simulator environment for TinyOS. With the addition of Misty1, TinySec library users now have an additional choice as regards the cipher.

Keywords: Sensor Network Security, Misty1, TinySec

ACKNOWLEDGEMENTS

I wish to thank my supervisor, Dr. Truman Yang, for not only introducing me to the interesting world of wireless sensor networks and the associated security sphere but also helping me to see how these concepts and challenges are tackled in practice (via the real examples of Crossbow motes, TinyOS, TinySec etc.). The initial guidance and direction by Prof. Yang allowed me to come up possibilities for my contribution to this topic, resulting in this project. Thank you to Mr. Naveen Sastry (co-author of TinySec) for his suggestions and replies to my concerns.

I am infinitely grateful to my parents and sister for their support and help, without which the going would have been much tougher.

I thank all my friends who have helped or been supportive to me. Finally, I also thank my former boss, Walter Pilar, for being understanding and supportive during the course of my program.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vi
LIST OF FIGURES	vii
LIST OF APPENDICES	viii
CHAPTER 1: Introduction	1
1.1: Security basics, WSN routing attacks and other security issues	2
1.2: List of acronyms	5
1.3: Cryptography basics	6
1.4: Project details and progression timeline	14
1.5: Report organization	15
CHAPTER 2: Misty1 Cipher	17
CHAPTER 3: TinySec and Misty1 nesC Implementation	20
3.1: TinySec	20
3.2: Misty1 nesC Implementation	26
CHAPTER 4: The ‘Listen’ tool and TOSSIM	30
4.1: The ‘Listen’ tool	30
4.2: TOSSIM (TinyOS Simulator)	31
CHAPTER 5: Testing Results of Misty1 Implementation	36
5.1: Testing	36
5.2: Understanding the simulator output	37
CHAPTER 6: Conclusion and Future Work	44
APPENDICES	46
REFERENCES	100

LIST OF TABLES

	Page
TABLE 1: TinySec, compatible transmitter and receiver types	23
TABLE 2: TinyOS, frame details	40

LIST OF FIGURES

	Page
FIGURE 1: Crossbow's MICAz, MICAz OEM version, MICA2DOT; Gateway/Interface Boards; SUN's SPOT (Small Programmable Object Technology)	1
FIGURE 2: Stream Cipher Operation	9
FIGURE 3: Block Cipher, ECB Mode of Operation	11
FIGURE 4: Block Cipher, CBC Mode of Operation	11
FIGURE 5: Block Cipher, CFB Mode of Operation	12
FIGURE 6: Block Cipher, OFB Mode of Operation	12
FIGURE 7: Block Cipher, CTR Mode of Operation	13
FIGURE 8: TinySec packet formats (a, b) and Standard TinyOS packet format (c)	25
FIGURE 9 (screenshot): Documentation generation for TinyOS based applications	27
FIGURE 10: TOSSIM architecture and the 'Listen' tool	35
FIGURE 11 (screenshot): Misty1 implementation test 2	37

LIST OF APPENDICES

	Page
APPENDIX A: Misty1M.nc, source code	46
APPENDIX B: crypto.h, source code	57
APPENDIX C.1: Makefile (Test1, nesC compiler), source code	59
APPENDIX C.2: TestM1A.h (Test1, nesC header file), source code	59
APPENDIX C.3: TestM1A.nc (Test1, configuration file), source code	60
APPENDIX C.4: TestM1AM.nc (Test1, module file), source code	61
APPENDIX D.1: Makefile (Test2, nesC compiler), source code	64
APPENDIX D.2: TestM1B.nc (Test2, configuration file), source code	64
APPENDIX D.3: TestM1BM.nc (Test2, module file), source code	65
APPENDIX E.1: Test1 TOSSIM/Simulation Output	69
APPENDIX E.2: Test1 'Listen' Tool Output	94
APPENDIX F: Test2 TOSSIM/Simulation Output	97

CHAPTER 1: Introduction

This project deals with security in Wireless Sensor Networks (WSN). Batteries are a common source of power and wireless communications is the method of choice for WSN. Being battery powered and to keep costs low, node's sensor hardware (including the microcontroller) and wireless communication protocols need to be thrifty and less demanding on its energy resource. WSN applications mainly include monitoring environments & tracking objects of interest. Examples include:

- Industry and Commerce
- Security and Protection
- Habitat, Climate and Structural Monitoring
- Military
- Healthcare

Some common examples of wireless sensor nodes include:

Figure 1 (clockwise from top left): Crossbow's MICAz, MICAz OEM version, MICA2DOT; Gateway/Interface Boards; SUN's SPOT (Small Programmable Object Technology)

As the technology and field of WSN matures towards wide-scale practical deployment, the requirement for reasonably secure communications is a reality that cannot be ignored. Without security features, adoption of WSN would be severely limited. Many real world applications require serious privacy protection mechanisms and unauthorized reading or malicious tampering of data is unacceptable. However, implementing security is not straight forward. This is because of a multitude of factors on the node and its network (typically characterized by ad-hoc network formation, node mobility, multi-hop transmissions and ability to support in-network data aggregation and clustering). Constraints include node energy, processing power (e.g. microcontroller and storage memory), low overhead due to security protocol, network scalability, network fault tolerance – node failure or link congestion adaptability, physical node capture including tampering or damage and wireless communications fundamentally being a “broadcast” method where everyone can “listen in”. Each one of these points impacts on potential use of existing security techniques. Thus we find that although the field of network security is well established, scaling down the existing robust techniques to the “realm” of sensor nodes is a challenge [1, 2, 3].

1.1 Security basics, WSN routing attacks and other security issues

Commonly identified secure communication requirements include [1, 2, 3]:

- **Data Confidentiality:** The aim is that data packet contents should be understood only by intended/authorized parties. This is usually addressed through the process of message encryption (secretly coding a message using a key), which is later decrypted (deciphered using a corresponding key) by authorized node(s).
- **Data Integrity:** The aim is detect if data has been corrupted (unintentional) or altered in transit (intentional). Common examples of mathematical tools used to detect data changes include Cyclic Redundancy Checks (CRC codes e.g. 16-bit CRC) or Cryptographic Hash Functions (e.g. Message Digest Function MD5 and SHA-1). The outputs of these mathematical functions are called CRC and hash respectively. For security applications, cryptographic hash functions should be used since CRC (or any of the other simpler functions such as parity, checksum etc) are not designed to resist attacks (intentional manipulation), rather they are useful when data may be disturbed by random processes

(such as communication channel noise). As of 2005, it has been reported on the Internet that both MD5 and SHA-1 have security flaws. Another method used to achieve data integrity (along with authentication) is called a MAC/MIC (Message Authentication/Integrity Code) which additionally uses a secret key as its input during the MAC/MIC computation.

- **Data Authentication:** The aim is to allow a receiver to verify that the received data was indeed sent by the claimed sender. This is useful for example in a situation where the base station (BS) wishes to re-program the WSN through a broadcast, however the nodes must first make sure that the new code image being sent down is in fact by the legitimate BS. Message Authentication Codes (also abbreviated as MAC – with respect to the security topics, while in networking topics MAC usually refers to Media Access Control) can be employed to satisfy combined needs of Integrity and Authentication. The MAC computed over a data packet is appended to the encrypted data. When dealing with authenticated “broadcast”, an interesting problem is that of several nodes sharing a global or common key (as is the case in symmetric cryptography), potentially allowing any “rogue” node to impersonate another valid node possibly even the BS – which is naturally undesirable.
- **Data Freshness:** The aim is to have a scheme that allows one to know if data is recent or old. It is useful against “replay” and “cut-paste” attacks. Time stamping is used at times. A simpler approach may be to use some sort of a counter/sequence number to implement order checking for packets being transferred in the network.
- **Semantic Security:** The aim is add variation in the ciphertext when there is little or no change in the corresponding plaintext. This is done to prevent an adversary from deducing the linkage between pairs of plaintexts-ciphertexts. If one is able to deduce the same (this may not be too difficult to achieve especially for low entropy periodic messages such as simple “Yes/No” type responses by nodes) then an adversary can simply replay the ciphertext heard earlier at a later time (assuming the encryption technique does not incorporate any sort of data “freshness” element). The variation is typically provided by using what is known as an initialization vector (IV). IVs can be random or counter type.

Examples of routing attacks (with a focus on WSNs, as opposed to the Internet or generic computer networks) include [1, 2, 3, 21, 25, 27]:

- **Black Hole Routing (Sink Hole Routing):** A sensor node converges all network traffic toward itself by continually advertising shorter “better quality” routes to the base station or all other nodes in the system. Once traffic flows towards it, it may then silently drop or modify packets.
- **Routing Loops (Sleep Deprivation Torture):** The motive of the attacker is to deplete the power source (e.g. battery) for a particular set of nodes. If carried out systematically, it may allow an attacker to “kill” a critical path, which may be important from network connectivity point of view. This could also be orchestrated so as to lead the original network towards being left over as a bunch of disjointed sets, unable to function as a whole.
- **Routing Table Overflow:** The idea is to advertise a large number of routes to non-existent nodes thereby causing routing table overflows, overwhelming and possibly leading to failure of the routing system.
- **Disclosing Network Topology:** Source routing protocols can be targeted by an attacker to map out/construct the network topology by looking at the list of routing nodes contained in the headers of packets responsible for network routing.
- **Denial of service (DOS) class attacks.**

Some algorithms are designed with a feature wherein a node passively listens to its own transmission which is intended to be forwarded by a neighbor. If it finds that the neighbor node silently drops or modifies the packet, it raises an alarm and considers the neighbor a “rogue” node. However, this scheme can be defeated by a pair of colluding nodes. Consider that node A transmits. Next “rogue” node X forwards the packet as a regular node would. However, node X’s neighbor, also a “rogue”, node Y drops/alters the packet.

To complete the security sphere there are other considerations such as physical capture, tampering or even damage to a node rendering it worthless. Tampering includes both cases, a) where the node may be reprogrammed as “rogue” node to transmit misleading information in the network and/or b) where sensitive & important information may be extracted from the node.

Suggestions have been made in literature that device tampering should cause the node to “flush” or clear its sensitive contents.

Interestingly we need to achieve all the above, while keeping the impact (overhead) of security features minimal and ensuring that security is easy to implement (for developers) or utilize (for end-users).

1.2 List of acronyms

WSN: Wireless Sensor Network

PT: Plaintext (Cleartext)

CT: Ciphertext

CRC: Cyclic Redundancy Check

MD[n]: Message Digest

SHA-[n]: Secure Hash Algorithm

MAC/MIC (field of cryptography): Message Authentication/Integrity Code

MAC (field of networking): Media Access Control

BS: Base Station

IV: Initialization Vector

DOS: Denial Of Service

HMAC: Hash Message Authentication Code

OMAC: One-key Message Authentication Code

PMAC: Parallelizable Message Authentication Code

CBC-MAC: Cipher Block Chaining Message Authentication Code

DH: Diffie-Hellman(-Merkle)

RSA: Rivest-Shamir-Adleman

RC[n] (cipher): Rivest Cipher

SEAL (cipher): Software-Optimized Encryption Algorithm

AES (cipher): Advanced Encryption Standard

DES (cipher): Data Encryption Standard

3DES (cipher): Triple DES

IDEA (cipher): International Data Encryption Algorithm

MISTY1 (cipher): Mitsubishi Improved Security Technology (or initials of the cipher designers Matsui Mitsuru, Ichikawa Tetsuya, Sorimachi Toru, Tokita Toshio, and Yamagishi Atsuhiro)

TEA (cipher): Tiny Encryption Algorithm

ECB (mode): Electronic Codebook

CBC (mode): Cipher Block Chaining

OFB (mode): Output Feedback

CFB (mode): Cipher Feedback

CTR (mode): Counter

OCB (mode): Offset Codebook

CCM (mode): Counter with CBC-MAC

GCM (mode): Galois/Counter Mode

CTS: Cipher Text Stealing

NESSIE (cryptographic research project, Europe): New European Schemes for Signatures, Integrity and Encryption

CRYPTREC (cryptographic research project, Japan): Cryptography Research and Evaluation Committee

SSH: Secure Shell

SSL: Secure Sockets Layer

S-box: Substitution box

P-box: Permutation box

1.3 Cryptography basics

Cryptography: Science which deals with the study of analyzing and developing techniques to encipher (encrypt) and decipher (decrypt) messages

Plaintext (PT): Information used as input to an encryption algorithm. In some systems that employ multi-layer encryption, the ciphertext from previous encryption process may become plaintext for the next process.

Ciphertext (CT): The output of an encryption algorithm. A message written in a secret manner intended to prevent unauthorized parties from knowing the contents of the message.

Key (Cryptovvariable): Sequence of numbers (typically long) which is kept secret. Typical key sizes include 40, 56, 64, 80, 128, 192, 256, 512, 1024, 2048 and even 4096 bits. As of 2006, 80 bit key is generally regarded as the minimum key length needed to prevent brute force attacks.

Ciphers: Mathematical functions which perform the process of encryption-decryption.

Hash: A repeatable method of converting data representing any sort of information (message) into a considerably shorter and generally fixed sized data (message digest), also called hash sum that basically serves as a digital “fingerprint” of the original data. Normally, the focus is on designing methods that yield minimum hash collisions (i.e. two messages yielding the same hash sums) in expected input domains. Hash functions with a “strong mixing” property are capable of producing large changes in the hash sums when small changes occur in the message. This is known as the avalanche effect. Cryptographic hash functions (used to detect purposeful data tampering by adversaries, in addition to transmission induced errors), when well designed, yield purely one-way operations (data source -> hash sums). SHA-1/MD5 are examples of hashing methods originally intended for cryptographic purposes, however as of 2005 they are not recommended for such use anymore due to weaknesses discovered. It is important to realize that typical error detection/correction (checksums and CRCs) focus on data corruption due to random processes (e.g. channel noise) and they can be easily exploited/manipulated by adversaries. Due to their vulnerability to attacks they should not be used for security implementations. MAC/MIC (Message Authentication/Integrity Code) additionally uses a cryptographic key input to generate the MAC/MIC tag. Block/stream ciphers and a construction known as Merkle-Damgard can be used to make hash functions. MAC functions can be made from cryptographic hash functions (e.g. HMAC) or, as they are often built, from block cipher operations (e.g. OMAC, PMAC and CBC-MAC).

Key management/exchange/agreement protocol: Helps in the secure establishment (generation and distribution) and maintenance (replacement at expiry) of key(s) between the two parties

wishing to carry out secure communications over a fundamentally insecure channel (for example wireless medium). Examples: Diffie-Hellman (DH helps two parties to jointly establish a shared secret key over an insecure communication channel), RSA (helps compute public-private key pair for one party) etc.

Public-key (form of asymmetric) cryptography: Uses a pair of keys (known as public-key and private-key). Messages encrypted with one key can only be decrypted with the corresponding 2nd key from the pair. For example if the public-key of the receiver is used to encrypt by the sender, only the private-key held by the receiver can be used to decrypt the ciphertext (this function is used for achieving confidentiality). On the other hand, if the private-key of the sender is used to encrypt, only the corresponding public-key (of the sender) can be used by anyone to decrypt the message (this function is used for implementing digital signatures – which in turn is used to provide sender authentication). In practice the message (plaintext if confidentiality is not required or ciphertext if it is required) is hashed and then the hash is digitally signed.

A potential weakness in public key cryptography methods is the possibility of what is known as “Man-in-the-middle attack”. In this attack, the communication of a public key request is intercepted by a third party. The third party then provides its own public key to the requesting party instead. In order for this attack to succeed the encrypted response must also be intercepted, decrypted and re-encrypted using the original/true public key and finally forwarded on. This way the middle party can be illegally listening to a communication otherwise intended to be secure. For example, the original DH exchange algorithm by itself does not provide authentication of the parties and is thus vulnerable to this attack. Authentication is generally needed to overcome this issue.

Symmetric key (secret-key, single-key, one-key or sometimes called private-key) cryptography: The same key (or trivially related keys i.e. simple transformation of one key yields the other) is used for both encryption and decryption purposes (the keys are pre-established and shared by the sender/receiver). Symmetric key ciphers can be broadly divided into two types, depending on whether they work on fixed-length groups of bits (block cipher) or on a continuous stream of symbols – bits or bytes (stream cipher).

Examples of symmetric encryption techniques include RC4, SEAL etc. (stream ciphers) and AES, Blowfish, DES, 3DES, IDEA, KASUMI, MISTY1, RC5, RC6, Twofish, TEA, Serpent, Skipjack (block ciphers) etc. For block ciphers, 64-bit blocks were used earlier, although some newer ciphers (e.g. AES aka Rijndael) use 128-bit blocks or even support variable block size. Typically symmetric key algorithms are considered more suitable (compared to public-key methods) for resource constrained systems (example: execution speed); however key management may be more difficult for them.

Stream ciphers are classified into synchronous and self-synchronizing stream ciphers. Synchronous stream ciphers are susceptible to attacks where flipping a bit in the ciphertext causes the same bit to be flipped in the plaintext. An example of self-synchronizing stream cipher is a block cipher in cipher-feedback mode (CFB).

Figure 2: Stream Cipher Operation

A typical stream cipher may use a small and convenient key size (128-bit) and then generate a pseudorandom keystream (not truly random) which can be combined with the plaintext. Stream ciphers suffer from a weakness that if the initialization vector is used twice, it is often possible to recover both plaintexts. Another important point is that the period of the keystream needs to be sufficiently large. If the sequence repeats then techniques exist that could allow the plaintext to be extracted. However, fast stream ciphers tend to be faster than block ciphers and have lower hardware complexity [3].

Block ciphers usually work with 8 or 16 bytes (fixed size) and provide unvarying transformation (stream ciphers provide varying transformation). The transformation is controlled by the key. To encrypt messages longer than the block size a mode of operation is used (note: certain modes of

operation cause the block cipher to effectively act as a stream cipher). If the plaintext is longer than for a k-byte block cipher the plaintext is broken into 'k' byte sized chunks, padded if necessary and then operated on. Block ciphers cannot directly work on data shorter than their block size. They are constructed by repeatedly applying a simpler function with each iteration being called a round and the corresponding function called a round function. Usage of 4 to 32 rounds is typical. Block ciphers commonly use the Feistel structure (combination of substitution and transposition techniques).

Block ciphers modes of operation include earlier modes such as ECB (Electronic Code Book), CBC (Cipher Block Chaining), OFB (Output Feedback) and CFB (Cipher Feedback) - all these only provide confidentiality and no integrity checking. Authenticated encryption (simultaneous encryption and MAC tagging) built using block ciphers, provide integrity and authentication check due to the MAC tag. Examples include CCM, EAX, GCM and OCB modes. Some other modes also exist (LRW, CMC and EME). Each mode has different characteristics with regard to the error propagation, ease of random access and vulnerability to certain types of attacks. All modes (except ECB) require an IV – a sort of initial data block to aid the start of the encryption process by providing randomization. In fact, ECB mode provides unvarying transformation and it is often possible to successfully analyze ciphertexts (especially with simple messages) and thus not recommended for practical use. In most cases IVs are not secret and are in fact transmitted in the clear to the receiver along with the ciphertext since the IV is required for decryption purposes. IV repetition with the same key is an issue with varying repercussions and hence should not be allowed (i.e. keys should be set to expire and replaced as necessary). For example CBC/CFB modes leak partial information about the plaintexts when IVs repeat (with same key), whereas OFB/CTR modes may allow both plaintexts to be recovered – something definitely undesirable [3].

Note that stream ciphers can also be built using block ciphers. For example block ciphers in OFB, CFB and CTR (Counter) modes effectively act as stream ciphers. These modes have the advantage of being able to utilize the same function for encryption and decryption (code re-use) – very desirable for resource constrained environments. However as mentioned above, they also share the same weaknesses as any stream cipher.

Electronic Codebook (ECB) mode encryption

Electronic Codebook (ECB) mode decryption

Figure 3: Block Cipher, ECB Mode of Operation

Cipher Block Chaining (CBC) mode encryption

Cipher Block Chaining (CBC) mode decryption

Figure 4 (above): Block Cipher, CBC Mode of Operation

Figure 5 (above): Block Cipher, CFB Mode of Operation

Figure 6 (above): Block Cipher, OFB Mode of Operation

Figure 7 (above): Block Cipher, CTR Mode of Operation

CBC mode is commonly used for block ciphers however its main drawback is that its execution cannot be parallelized. Also the message must be padded to a multiple of the block size or one must employ “ciphertext stealing” technique. CBC mode requires both encryption and decryption functions. Also it has been designed to be used with random IVs, and as such if counter type IVs are used then CBC mode suffers from leakage issues [3].

For resource constrained systems, in order to keep secure communication overhead and corresponding radio usage to a minimum (thereby reducing the strain on the power source) it is desirable that during the encryption process message expansion does not take place or if it does then it is kept to a minimum. In other words, the size of ciphertext equals or is only slightly more than the size of corresponding plaintext. Zero message expansion (i.e. size of CT = size of PT) can be achieved using a technique known as “ciphertext stealing”. One advantage with block ciphers is that the corresponding MAC functions tend to use the same function as the one used in the block ciphering operation. In general, given the constraints of program and data memory on nodes, code re-use is advantageous. Note that the same key should not be used for encryption and MAC calculation purposes [3].

Hybrid key cryptography: Mixed use of public-key and symmetric cryptography. For example when sending a message, the sender encrypts the message with a symmetric key and includes the symmetric key, in encrypted form, along with the CT (the key is encrypted using a public-key method e.g. RSA). Thus only a limited amount of information is encrypted using the comparatively slower RSA method, whereas the main message is encrypted using the faster symmetric key algorithm. Most public-key algorithms are relatively computationally costly (compared to many symmetric key algorithms of roughly equivalent security).

1.4 Project details and progression timeline

During my Directed Studies course I familiarized myself with the following:

- Wireless Sensor Networks including commercial hardware available on the market
- Cryptography
- Read papers dealing with WSN security. These included a paper entitled ‘SPINS: Security protocols for Sensor Networks’ [1], a thesis entitled ‘A framework for implementing security in wireless sensor networks’ [2] and another paper entitled ‘TinySec: A link layer security architecture for wireless sensor networks’ [3, 4]. This is where I got introduced to TinySec.
- Rounded up the various simulation and research packages. Examples include EMStar, ns-2, SENSE, SENS, ATEMU, AVRORA, GloMoSim, OMNet++, TOSSIM [5-9].

Further during the first term of my project I read up and approached hands-on the various related topics such as TinyOS, nesC programming language [15, 16], TOSSIM+TinyViz [9, 10], TinySec. I also learnt how to use hardware MICAz motes (sensor nodes) with Crossbow’s MoteWorks platform. However, standard TinyOS was adopted later on since MoteWorks does not support TinySec. The next step I took was to learn how to integrate TinySec into an existing data monitoring application. Surge is a sample application included with a standard installation of TinyOS. I enabled TinySec within this application and simulated it using TOSSIM and a Java based tool called ‘Listen’ (also provided with the standard installation of TinyOS).

Having reached this point, I considered the following options towards extending my project work in a meaningful manner and one that would be my own contribution:

- Port (C -> nesC) new cipher into the TinySec implementation. Existing ciphers in TinySec included Skipjack and RC5.
- Extend TinySec to support newer hardware (micaZ). Currently TinySec supports mica2, mica2dot and TOSSIM simulator.
- Study and enable hardware based AES encryption available on newer micaZ motes.
- Implement some key management protocol on top of the global/network-wide shared, symmetric key model used by TinySec.

I decided to go with the first option. Initially I looked at AES but found out that it deals with data blocksize = 128-bit (16 byte) [22, 23, 24]. TinySec library has primarily been designed for and tested with 64-bit (8 byte) blocksize. As a result, I chose another cipher called MISTY1 [18, 28, 29, 30].

1.5 Report organization

- Chapter 1 (Introduction) introduces the reader to security in wireless sensor networks through the topics of security basics, WSN routing attacks and other security issues. It proceeds to briefly explain the various terms one is likely to encounter when dealing with cryptography and includes a list of acronyms. Finally project details and progression timeline is provided.
- Chapter 2 (Misty1 Cipher) provides key introductory details about the Misty1 cipher. It includes a link to RFC document on the Internet where complete details regarding the cipher can be found.
- Chapter 3 (TinySec and Misty1 nesC Implementation) introduces the reader to TinySec and summarizes its features, parameters plus basic usage. Next the new cipher's implementation process is explained and it's usage details within TinyOS are outlined.

- Chapter 4 (The ‘Listen’ tool and TOSSIM) gives some details regarding the tools (e.g. standard TinyOS java applications and the simulator) utilized for testing the new cipher code.
- Chapter 5 (Testing Results of Misty1 Implementation) provides details of the test approaches employed and includes information useful in the understanding the simulator output.
- Chapter 6 (Conclusion and Future Work) closes the report with concluding remarks.
- The appendices section contains all software code as well as simulation output listings.
- The references section lists all the references used in this project.

CHAPTER 2: Misty1 Cipher

MISTY1 is a block cipher originally designed in 1995 by Mitsuru Matsui, Ichikawa Tetsuya, Sorimachi Toru, Tokita Toshio, and Yamagishi Atsuhiko for Mitsubishi Electric. MISTY1 was one of the selected algorithms in the European NESSIE project, and has been recommended for Japanese government use by the CRYPTREC project. KASUMI is a strengthened version of the MISTY1 cipher and has been adopted as the standard encryption algorithm for European mobile phones. 'MISTY' can be interpreted to stand for 'Mitsubishi Improved Security Technology' or as simply the collection of the initials of the researchers involved in its development. MISTY1 is covered by patents, although the algorithm is freely available for academic (non-profit) use. Complete details on the cipher and its algorithm can be found at <http://tools.ietf.org/html/rfc2994>

It uses a key size of 128 bits and block size of 64 bits. It is based on a nested Feistel network and can be implemented as $4 \times n$ variable number of rounds (8 are recommended, i.e. $n = 2$ – balancing needs of execution speed and encryption strength). MISTY1 has an innovative recursive structure; the round function itself uses a 3-round Feistel network. Its design is intended to allow high-speed encryption on hardware platforms as well as on software environments, including low cost applications. For example the cipher's structure is suited to parallel computation implementation and allows one to achieve high performance. The look-up tables have been optimized to boost hardware performance. The choice of 128-bit key and 256-bit subkey takes into account security and practicality (performance). The design targets varied platforms such as 8-bit microprocessors/microcontrollers for smart card applications, 16-bit DSP for wireless communication systems to higher end 32-bit and 64-bit microprocessor based applications. As an example one software implementation of Misty1 on a typical 8-bit Z80 microprocessor uses just 1598 ROM bytes and 44 RAM bytes, so that one block encryption with key scheduling is executed in approximately 5 ms at 5MHz. On the hardware side an example is that of 800Mbps encryption speed achieved in approximately 50K gate design that encrypts one block of plaintext in one cycle. As of 2000 Misty1 is widely employed in several governmental and commercial projects.

MISTY1 claims to be provably secure against linear and differential cryptanalysis. It was carefully designed to withstand many cryptanalytic attacks known to the cryptographic community at time of its design. Higher order differential cryptanalysis, proposed by Lai and developed by Knudsen, is unlikely to be applied against a Misty1 implementation with full 8-rounds due to the inclusion of the FL function (function explained more below). Similarly the slide attack, invented by Biryukov and Wagner, and impossible differential cryptanalysis, proposed by Biham and Shamir, are likely protected against due the use of the FL function in the design. Interestingly, the slide attack was in fact discovered later than the design time of the cipher; however Misty1 stands up to it because one of its design principles was protection from related-key type attacks. Finally implementational attacks (e.g. timing and power analysis) are often realistic and powerful attacks against smart card applications. To guard against these an implementor could utilize established counter measures (e.g. removing 'jump' instructions and/or variable cycle instructions in software based implementations).

The algorithm consists of 2 main steps: key scheduling part and data randomizing part. The key scheduling part takes 128-bit input key (K) and produces a 128-bit expanded key (EK). Data randomizing part takes a 64-bit plaintext and generates the ciphertext (encryption process). If data randomizing part is processed in reverse order, ciphertext is transformed to plaintext (decryption process).

Data randomizing part uses two kinds of function, which are called function FO and function FL. Function FO calls another function, called FI. The key expansion part also uses function FI. Function FI uses two S-boxes, called S7 and S9.

Function FO takes two parameters. One is a 32-bit width input data, called FO_IN. The other is an index of EK, called k. It returns a 32-bit width data, called FO_OUT.

Function FI takes two parameters. One is a 16-bit width input data, called FI_IN. The other is a part of EK, called FI_KEY, which is also 16-bit width. It returns a 16-bit width data, called FI_OUT.

S7TABLE and S9TABLE denote the S-boxes S7 and S9 respectively in terms of look up table notation.

Functions FL and FL_INV take two parameters. One is a 32-bit data, called FL_IN. The other is an index of EK, called k. They return a 32-bit width data, called FL_OUT. When the algorithm is used for decryption, function FLINV is used instead of function FL.

By comparison note that Skipjack is a block cipher (symmetric key) algorithm which is also based on Feistel network. It uses 32 rounds, an 80-bit key and 64-bit block size [17].

CHAPTER 3: TinySec and Misty1 nesC Implementation

3.1 TinySec

TinySec is a generic security library that developers can integrate into TinyOS sensor network applications. It is portable to a variety of hardware and radio platforms. It can be used in testing and evaluating higher level security packages. The authors expected higher level security protocols (e.g. key distribution protocols, some possibly based on public key cryptography) to use TinySec in order to create secure pairwise communication between nodes. It is included with the standard TinyOS distribution. TinyOS can run on a host of processors manufactured by the likes of TI, Atmel, Intel x86, ARM and supports Chipcon CC1000 and RFM TR1000 radio architectures.

The authors made an effort to show that software based link layer protocols are indeed feasible and efficient (adding less than 10% energy, latency and bandwidth overhead in the authors' 36 node system evaluation implementation). TinySec is meant to cover the basic security needs of all but the most security critical applications. An interesting point that the authors made in their paper was that the security feature should be easy to implement (for developers and end-users) and minimally impact overall performance. They based their observations on lessons learnt from other technologies such as GSM, 802.11 wireless LAN standard, Bluetooth and IEEE 802.15.4. The authors mentioned 5 other projects using TinySec in standard or modified form in their paper [12, 13].

Summary of TinySec:

- For ease of development a network wide shared key model is used. The key is included at compile time (key file is maintained on the developer's computer). Thus, TinySec uses a single, shared/global, symmetric cryptographic key. The key file (`.tinyos_keyfile`) was located in `/home/user` inside my cygwin installation. The 128-bit key is in fact utilized as a pair of 64-bit keys, one each for encryption and MAC calculation purposes (note: a common 64-bit key should not be used for both purposes). The 64-bit encryption key is further expanded into an 80-bit key by the `setupKey()` function inside the cipher implementation i.e. 'SkipJackM.nc'.

- Runs on platforms such as Mica (RFM TR1000 radio) and Mica2, Mica2dot (both using Chipcon CC1000) and TOSSIM simulator.
- Link layer encryption and integrity protection - transparent to applications. Hop-by-hop security (v/s end-to-end security techniques like SSH/SSL, typically used in Internet) implies better support for in network data aggregation.
- Software implementation: 3000 lines of nesC code, requires 256 bytes of RAM, 8152 bytes of program space. The default TinyOS 1.1.2 radio stack was modified by the authors (to redirect byte level radio events to the TinySecM module). The authors also modified the scheduler into a two-priority scheduler (two queues).
- Uses cipher block chaining (CBC block cipher) encryption scheme with “ciphertext stealing” to avoid message expansion due to encryption and achieves code re-use due to the MAC and encryption processes sharing the same block cipher code.
- TinySec uses Skipjack cipher as default (since it has minimal key setup costs and is free from patent issues) or RC5 cipher; however TinySec is basically cipher-independent.
- Message Authentication & Integrity achieved using MAC. MAC is computed over the entire message (header i.e. IV + data payload. Note payload may be in PT form (clear) if using TinySec-Auth or CT form (encrypted) if using TinySec-AE). MACs help protect against data tampering (versus simple CRC checksums) in addition to detecting transmission errors. TinySec MAC field is 4 bytes, default maximum data payload = 29 bytes.
- TinySec overhead: Packet lengths increase by 1 byte (TinySec-Auth, from 36 to 37 bytes) or 5 bytes (TinySec-AE, from 36 to 41 bytes) over TinyOS. Note standard TinyOS uses a 36 byte packet. 2 options: (1) TinySec-AE (data payload is encrypted and authenticated with MAC computed over encrypted data + IV i.e. header) and (2) TinySec-Auth (complete packet i.e. unencrypted data payload + header is authenticated with MAC, but data payload is not encrypted)
- A changing Initialization Vector (IV) is used with each invocation of encryption process. The main purpose is to add variation to the encryption process when there is little variation in the plaintext (i.e. semantic security). IV is sent in the clear along with the encrypted data. TinySec relies on 8-byte (IV size), counter (not random) based IV composed of the following fields:

16-bit destination | 8-bit Active Message Handler type | 8-bit data payload length | 16-bit source | 16-bit counter

- The 16-bit counter starts at 0 and the sender increases it by 1 after each message is sent.
- Maximum data payload (property of TinyOS) = 29 bytes.
- Length byte (upper two bits) utilized by TinySec authors to carry information about whether the mode is standard TinyOS, TinySec-AE or TinySec-Auth.
- Note that IV re-use is only a problem when one re-uses the same IV with the same key. A key update protocol could be used to manage TinySec keys (expire/generate/distribute)
- The idea behind 'Active Message Handler type' is similar to ports in TCP/IP. It helps a node pass received data to the appropriate "handler" function. It can also be useful in saving energy by turning off the radio, via a feature called "early rejection", if a message is not applicable to the node.
- TinySec does not require any changes to the application code. To enable TinySec, the application writer only needs to specify 'TINYSEC=true' and 'GOALS += tinysec' on the command-line or in the makefile during the 'make' process. This enables TinySec-Auth (default selection). The application writer can then switch modes to TinySec-AE by making a function call in the code.
- TinyOS uses group IDs to differentiate between various sensor networks operating simultaneously. Group IDs are not used in TinySec; instead different networks should use different keys!

In order to integrate TinySec into one's application the basic modifications required include,

- **Inside *Configuration* (MyApp.nc) file:**

```
configuration MyApp {  
  }  
  implementation {  
    components TinySecC;  
    MyAppM.TinySecMode -> TinySecC.TinySecMode;  
  } // end: configuration implementation
```

- **Inside *Module* (MyAppM.nc) file:**

```
uses {  
  interface TinySecMode;  
}
```

```

implementation {
// Typically call the following commands inside StdControls.init()
call TinySecMode.setTransmitMode(uint8_t txmode);
// where txmode = TINYSEC_AUTH_ONLY or
//               TINYSEC_ENCRYPT_AND_AUTH or
//               TINYSEC_DISABLED
call TinySecMode.setReceiveMode(uint8_t rxmode);
// where rxmode = TINYSEC_RECEIVE_AUTHENTICATED or
//               TINYSEC_RECEIVE_CRC or
//               TINYSEC_RECEIVE_ANY
} // end: module implementation

```

Transmitter node type	Compatible receiver node type
DISABLED	RECEIVE_CRC
AUTH_ONLY, ENCRYPT_AND_AUTH	RECEIVE_AUTHENTICATED
AUTH_ONLY, ENCRYPT_AND_AUTH, DISABLED	RECEIVE_ANY

Table 1: TinySec, compatible transmitter and receiver types

Refer to <http://tinyos.cvs.sourceforge.net/tinyos/tinyos-1.x/apps/TestTinySec/> for further understanding how a basic TinySec app may be coded. Refer to `/tos/interfaces` directory and open up any TinySec *interface* file (TinySecControl, TinySecMode, TinySecApp, TinySec, TinySecRadio) to find out all supported *commands/events*.

To change the cipher used by TinySec, open up `/tos/lib/TinySec/TinySecC.nc` *configuration* file.

- **Inside *Configuration* (TinySecC.nc) file:**

```

configuration TinySecC {
...
...
...
}

implementation {
components SkipJackM as Cipher;

```

```
// change to: RC5M as Cipher or Misty1M as Cipher
} // end: configuration implementation
```

Open up /tos/lib/TinySec/TinySecAppC.nc *configuration* file.

- **Inside *Configuration* (TinySecAppC.nc) file:**

```
configuration TinySecAppC {
...
...
...
}
implementation {
components SkipJackM;
// change to: RC5M or Misty1M

CBCModeM.BlockCipher -> SkipJackM.BlockCipher;
CBCModeM.BlockCipherInfo -> SkipJackM.BlockCipherInfo;
// change to:
// CBCModeM.BlockCipher -> RC5M.BlockCipher;
// CBCModeM.BlockCipherInfo -> RC5M.BlockCipherInfo;
// or
// CBCModeM.BlockCipher -> Misty1M.BlockCipher;
// CBCModeM.BlockCipherInfo -> Misty1M.BlockCipherInfo;
} // end: configuration implementation
```

Open up /tos/lib/TinySec/crypto.h *header* file.

- **Inside header (crypto.h) file: change the below code,**

```
typedef struct CipherContext {
    // rc5      needs 104 bytes
    // skipjack needs 20 bytes
    uint8_t context[20];
} CipherContext;
```

To the following if using RC5 cipher,

```
typedef struct CipherContext {
    // rc5      needs 104 bytes
    // skipjack needs 20 bytes
```

```
uint8_t context[104];
} CipherContext;
```

Or to the following if using Misty1 cipher,

```
typedef struct CipherContext {
    // rc5      needs 104 bytes
    // skipjack needs 20 bytes
    // misty1   needs 64 bytes
    uint16_t context[32];
} CipherContext;
```

(a) TinySec-AE packet format

(b) TinySec-Auth packet format

(c) TinyOS packet format

Figure 8: TinySec packet formats (a, b) and Standard TinyOS (classic) packet format (c). Values in brackets are number of bytes. Hatched fields are protected by the MAC. The shaded/grey field (i.e. data field for TinySec-AE mode) is encrypted [3]

Above diagram compares the standard `TOS_Msg` with the structures used by TinySec:

Total size: 36 bytes where TinyOS Header = 5 bytes, TinyOS Data Payload = 29 bytes (max, can be less) and TinyOS CRC-16 = 2 bytes. Note that TinySec packets contain neither Group ID (TinySec uses different keys instead) nor CRC-16 (CRC is replaced by MAC in TinySec) fields!

3.2 Misty1 nesC Implementation

TinySec code is built in a modular fashion so that the different sub-components can be replaced with ease. For example the cipher implementations can be found inside the files ‘SkipJackM.nc’ and ‘RC5M.nc’ located in the TinySec library. These modules provide what are called *interfaces* (BlockCipher and BlockCipherInfo) to their callers (module files ‘CBCMAC.nc’ and ‘CBCModeM.nc’). When replacing *modules* (e.g. the cipher implementation) as long as the *interfaces* and corresponding *commands* required by the callers are supplied by the new implementation, the cipher is essentially an independent entity.

The details of what these *interfaces* are and the *commands* to be implemented can be found out in two possible ways. One of them is to directly open up the cipher *module* file (e.g. SkipJackM.nc). The other way is to generate the *component* structure documentation (which includes a graphical representation of the *component* relationships within an application). TinyOS source files include metadata within comment blocks that ncc (the nesC compiler) uses to generate html formatted documentation.

To generate the documentation, use the following command inside a Cygwin window (after navigating to the TinySec enabled application directory):

```
make <platform> docs
```

where platform = mica2, micaz, pc etc. (pc should be used when dealing with TOSSIM). The resulting documentation can be accessed at the following location (An example screenshot follows): /tinyos-1.x/doc/nesdoc/<platform>index.html

Figure 9 (screenshot): Documentation generation for TinyOS based applications

Following is a list of *commands* implemented within the cipher *module* file:

- This command returns the preferred block size that this cipher operates with. It is always safe to call this function before the `init()` call has been made.

```
async command uint8_t BlockCipherInfo.getPreferredBlockSize()
```

Returns - preferred block size for this cipher. In the case where the cipher operates with multiple block sizes, this will pick one particular size (deterministically).

- This command initializes the BlockCipher context.

```
command result_t BlockCipher.init(CipherContext * context, uint8_t
blockSize, uint8_t keySize, uint8_t * key)
```

Parameter: context - structure to hold the opaque data from this initialization call. It should be passed to future invocations of this module which use this particular key.

Parameter: blockSize - size of the block in bytes. Some cipher implementations may support multiple block sizes, in which case any valid size is valid.

Parameter: keySize - key size in bytes

Parameter: key - pointer to the key

Returns - Whether initialization was successful. The command may be unsuccessful if the key size or blockSize are not valid for the given cipher implementation.

- This command encrypts a single block (of blockSize) using the key in the keySize.

```
async  command  result_t  BlockCipher.encrypt(CipherContext  *context,
uint8_t *plainBlock, uint8_t * cipherBlock)
```

Parameter: context - holds the module specific opaque data related to the key (perhaps key expansions).

Parameter: plainBlock - a plaintext block of blockSize

Parameter: cipherBlock - the resulting ciphertext block of blockSize

Returns - Whether the encryption was successful. Possible failure reasons include not calling init().

- This commands decrypts a single block (of blockSize) using the key in the keySize. Not all ciphers will implement this function (since providing encryption is a useful primitive).

```
async  command  result_t  BlockCipher.decrypt(CipherContext  *context,
uint8_t *cipherBlock, uint8_t *plainBlock)
```

Parameter: context - holds the module specific opaque data related to the key (perhaps key expansions).

Parameter: cipherBlock - a ciphertext block of blockSize

Parameter: plainBlock - the resulting plaintext block of blockSize

Returns - Whether the decryption was successful. Possible failure reasons include not calling init() or an unimplemented decrypt function.

Any code (S-boxes, functions FO/FI/FL/FL_INV, function setupKey(), macros s2cM/c2sM etc) besides the above is solely internal to the cipher implementation and can be changed (e.g. optimized) without causing problems to the way TinySec library works.

The original C source code version was written up by Hironobu Suzuki and can be found at the following location (<http://www.pp.iij4u.or.jp/~h2np/archives/misty1-gcc2.95.tar.gz>) [19]. This source code was ported to nesC, modified to support *interfaces*, *commands*, *events* etc

(compatible with current TinySec library) and tested by me as part of my project. Some important usage details:

- In file 'crypto.h', define struct CipherContext as
uint16_t context[32];
instead of
uint8_t context[20];
- Ensure wiring is changed in files: 'TinySecC.nc' and 'TinySecAppC.nc'
In 'TinySecC.nc' list the component
Misty1M as Cipher
instead of
SkipJackM as Cipher

In 'TinySecAppC.nc' list the *component* Misty1M instead of SkipJackM
& wire
CBCModeM.BlockCipher -> Misty1M.BlockCipher;
CBCModeM.BlockCipherInfo -> Misty1M.BlockCipherInfo;
instead of
CBCModeM.BlockCipher -> SkipJackM.BlockCipher;
CBCModeM.BlockCipherInfo -> SkipJackM.BlockCipherInfo;

CHAPTER 4: The ‘Listen’ tool and TOSSIM

4.1 The ‘Listen’ tool

Several PC application tools that are supplied with and work in conjunction with TinyOS networks are coded in Java. At the lower levels, there is piece of software called javax.comm package which helps Java apps use the serial port (hardware). This package is installed by default with a standard TinyOS installation (e.g. CVS snapshot release ver 1.1.15).

The ‘Listen’ program is the most basic way of communicating with the mote; it directly opens the serial port and just dumps packets to the screen.

Open a cygwin window, navigate to /tools/java and type,
`export MOTECOM=serial@serialport:baudrate`

The environment variable MOTECOM tells the ‘Listen’ tool (and most other tools too) which packets it should listen to. Thus `serial@serialport:baudrate` instructs the tool to listen to a mote connected to a serial port, where `serialport` is the serial port that you have connected the programming board to, and `baudrate` is the specific baudrate of the mote. For the mica and mica2dot motes, the serial port baud rate is 19200, for the mica2 it is 57600 baud. You can also use a mote name as the baudrate (in which case the mote’s standard baud rate is selected).

Examples:

```
export MOTECOM=serial@COM1:19200      # mica baud rate
export MOTECOM=serial@COM1:mica        # mica baud rate, same as 1st export
MOTECOM=serial@COM2:mica2              # the mica2 baud rate on COM2
export MOTECOM=serial@COM3:57600       # explicit mica2 baud rate
export MOTECOM=serial@COM1:micaz       # micaz baud rate
export MOTECOM=tossim-serial           # TOSSIM (virtual) serial port
```

Next type,

```
java net.tinyos.tools.Listen
```

The app will simply print the raw data of the packets received on the serial port to the screen.

```
unset MOTECOM
```

Use the above command to avoid forcing all Java applications to use the serial port to get packets. The `/tools/java/net/tinyos` directory contains many useful Java apps.

4.2 TOSSIM (TinyOS Simulator)

TOSSIM simulator executables build directly from TinyOS code. Instead of compiling a TinyOS application for a mote, users can compile it into the TOSSIM framework and run it on a PC. This allows one to use debuggers (e.g. GDB) and other traditional development tools. One limitation to be kept in mind: Spin locks or task locks in TinyOS apps will never exit in TOSSIM! TOSSIM simulates the 40Kbit RFM (radio) mica networking stack. It does not simulate the mica2 Chipcon CC1000 stack.

Compiling and Running a Simulation:

Open a cygwin window, navigate to the desired app's directory and type,

```
make
```

The above will compile the application and place the various executables (for different platforms) inside a sub-directory called 'build'. Among these, the one of our interest currently is a subdirectory called 'pc'. Within it, the TOSSIM executable is named 'main.exe'.

If you wish to compile only the TOSSIM executable, one can also type:

```
make pc
```

Inside a cygwin window, next navigate to `/apps/(yourapp)/build/pc` and type,

```
main -h
```

This will display a help message indicating all available options. Some immediately useful options include `-l` (scale simulation time), `-t` (run simulation for x virtual seconds) and `num_nodes` (number of nodes to simulate – compile time upper bound is specified in `/apps/Makerules`; standard TinyOS distribution sets this at 1000). By default all nodes boot in the first 10 seconds of simulation, with boot up times uniformly distributed.

TOSSIM provides control of the debug output desired at simulation run-time. TinyOS sources (*module files*) contain corresponding debug statements.

E.g. Standard TinyOS files, such as `tos/system/LedsC.nc`

```
async command result_t Leds.init() {
    atomic {
        ledsOn = 0;
        dbg(DBG_BOOT, "LEDS: initialized.\n");
        TOSH_MAKE_RED_LED_OUTPUT();
        TOSH_MAKE_YELLOW_LED_OUTPUT();
        TOSH_MAKE_GREEN_LED_OUTPUT();
        TOSH_SET_RED_LED_PIN();
        TOSH_SET_YELLOW_LED_PIN();
        TOSH_SET_GREEN_LED_PIN();
    }
    return SUCCESS;
}
```

```
async command result_t Leds.redOn() {
    dbg(DBG_LED, "LEDS: Red on.\n");
    atomic {
        TOSH_CLR_RED_LED_PIN();
        ledsOn |= RED_BIT;
    }
    return SUCCESS;
}
```

Another standard TinyOS file example, `tos/system/UARTM.nc`

```
command result_t Control.init() {
    dbg(DBG_BOOT, "UART initialized\n");
    atomic {
        state = FALSE;
    }
    return SUCCESS;
}
```

Similarly the user's app can be embedded with such `dbg()` statements by the programmer. Notice that each debugging statement is accompanied by one or more modal flags. When TOSSIM starts it reads in the cygwin environment variable `DBG` to determine which modes

should be enabled. In other words the DBG environment variable helps us to filter the messages being announced by TOSSIM. By default TOSSIM will announce all messages; however this generates a lot of information and is difficult, possibly even detrimental to our analysis/debugging efforts. For example, one of the common messages announced will be the bit-level events related to the radio sub-system since it operates at 40 Kbps for data tx/rx, 20 Kbps for receiving 'START' symbol (media access control layer; in any receiver mote) and 10 Kbps for sending 'START' symbol (media access control layer; in the transmitter mote). If desired, multiple debug modes can be enabled simultaneously. The set of DBG modes recognized by TOSSIM is printed with the help message. Inside a cygwin window, navigate to /apps/(yourapp)/build/pc and type,

```
main -h
```

Now look near the bottom of the help message. Notice: crypto = Cryptographic operations (e.g. TinySec) debug mode. Debugging modes include:

```
Known dbg modes: all, boot, clock, task, sched, sensor, led, route, am, crc,
packet, encode, radio, logger, adc, i2c, uart, prog, sim, queue, simradio,
hardware, simmem, usr1, usr2, usr3, temp, error, none
```

Note that if you don't set the cygwin environment variable DBG then all debug output is enabled in TOSSIM by default and the simulation will run very slowly.

Typical nesC dbg statement inside your app's source would look like,

```
dbg(DBG_USR1, "Data packet \%x processed\n", dpckt);
```

If you set DBG (cygwin environment variable) to 'usr1', i.e. export DBG=usr1 and run your app's simulation, the above debug statement (when executed in your app) will be announced by TOSSIM.

The following is also valid app code,

```
dbg(DBG_USR1|DBG_AM, "Data packet \%x processed\n", dpckt);
```

Notice that generally the name of the mode flag in the TinyOS source is 'DBG_' followed by the name of the DBG mode in uppercase.

Now if you set `DBG` (cygwin environment variable) to `'usr1'` or `'am'` or `'usr1,am'` and run your simulation, the above debug statement, when executed in your app, will be announced by TOSSIM. Comma-delimited strings are allowed when specifying the debug modes, for example:

```
export DBG=usr1,am
```

TOSSIM provides a set of communication services for interacting with external applications. These services allow programs to connect to TOSSIM over a TCP socket to monitor or actuate a running simulation. To view the data being sent to the serial port (e.g. the virtual serial port *'tossim-serial'*), one can use the *'Listen'* tool. The tool's input source must be set correctly as explained earlier in this document. Additionally, an external program can bi-directionally interact (using an intermediate tool called *SerialForwarder*) with the simulated wireless sensor network via the *tossim-serial* mode (virtual serial port) or the *tossim-radio* mode. In case of the *tossim-serial* mode, the interaction is via the simulated mote 0's virtual serial port connection.

TinyViz is a Java visualization and actuation environment for TOSSIM. It is a "framework" and by itself does little. Most of the functions are done by what are called 'plugins' (tools/java/net/tinyos/sim/plugins). Inside the TinyViz Java app each plugin is displayed as a pane (tab) on the right hand side. The tabs need to be activated by clicking on the menu item 'Plugins' located in the top left corner of the TinyViz app.

One example to start up a TinyOS app simulation is outlined below:

- Open a cygwin window, navigate to the desired app's directory and type,

```
make pc
```
- Set `DBG` (cygwin environment variable) as desired, for example:

```
export DBG=boot,led,am,uart
```
- Navigate to `/apps/(yourapp)/build/pc` and type,

```
./main -l=1 num_nodes
```

After the simulator boots (which may take a few moments in Cygwin), you'll see a lot of debugging information stream by. Typing control-C will stop the simulation.

Time is represented as hours:minutes:seconds. The zero preceding each line indicates which mote is the source of the debugging statement. In the simulator, motes start one after another, spaced by a short random period; this is to prevent perfect synchronization of motes.

Figure 10: TOSSIM architecture and the 'Listen' tool

CHAPTER 5: Testing Results of Misty1 Implementation

5.1 Testing

The new cipher implementation (Misty1M.nc) was tested using two approaches. 1st approach (TestM1A) was to simulate multihop routing [14] based radio transmission of test data in a simulated WSN and finally reception of the same test data via the virtual serial port (*tossim-serial*). Since node ID 0 decrypts data before sending it to the virtual serial port, if the data displayed by the Java tool `net.tinyos.tools.Listen` was correct we could verify that the received data was indeed being decrypted correctly by the new cipher code. The encrypted/decrypted bytes and MAC tags can be viewed in the TOSSIM simulator output during the radio transmit event (listings E.1 and E.2 in the appendices are related simulation outputs)

The 2nd test (TestM1B) I did was to use application level encryption/decryption (i.e. the data is not transmitted/received via radio or virtual serial port respectively). Some test data (held in one buffer) is encrypted in the test application and then decrypted to another buffer. Next the 1st buffer is compared with the 2nd buffer to see if the encryption-decryption process is working correctly (Below is a screenshot of the 2nd test and listing F in the appendices is the corresponding TOSSIM simulation output)

```

/opt/tinyos-1.x/apps/TestMisty1B/build/pc
$ main 1
SIM: Random seed is 66760
SIM: Time for mote 0 initialized to 6945579.
0: Cipher: Misty1
0: k[0] = 1
0: k[1] = 23
0: k[2] = 45
0: k[3] = 67
0: k[4] = 89
0: k[5] = ab
0: k[6] = cd
0: k[7] = ef
0: k[8] = 1
0: k[9] = 23
0: k[10] = 45
0: k[11] = 67
0: k[12] = 89
0: k[13] = ab
0: k[14] = cd
0: k[15] = ef
0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0:
0xef 0:
0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:
0: TestM1BM: IV = 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:
0: TestM1BM: CIPHERTEXT = 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0:
0: TestM1BM: RECOVERED P-TEXT (pre-enc) = 0: A 0: A 0: A 0: A 0: A 0: A 0:
A 0:
0: TestM1BM: Timer fired
0: TestM1BM: Proceeding to encrypt data
0: TestM1BM: Data encryption done
0: TestM1BM: CIPHERTEXT (post-enc) = 0: - 0: _ 0: X 0: S 0: c 0: n 0: A 0: 4 0:
0: TestM1BM: Proceeding to decrypt data
0: TestM1BM: Data decryption done
0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0:
0xef 0:
0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:
0: TestM1BM: IV = 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:
0: TestM1BM: CIPHERTEXT = 0: - 0: _ 0: X 0: S 0: c 0: n 0: A 0: 4 0:
0: TestM1BM: RECOVERED P-TEXT (post-dec) = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0:
! 0:
Exiting on SIGINT at 1:4:26.07695175.
/opt/tinyos-1.x/apps/TestMisty1B/build/pc
$

```

Figure 11 (screenshot): Misty1 implementation test 2

5.2 Understanding the simulator output

In order to understand the packets being transmitted (either via radio or the virtual serial port), the following information will be useful (note that byte stream format is little endian) [26]:

- When data (non-TinySec) is transmitted by a node the frame payload format is as follows:
Frame payload = TOS_Msg header (5 bytes) + TOS_Msg payload (typically max 29 bytes*)
+ CRC-16 (2 bytes)

* unless TOSH_DATA_LENGTH is defined to be something different in your app. The default is defined in `tos/types/AM.h` and is set to a value of 29.

TOS_Msg header fields:

Destination: 2 bytes

AM type: 1 byte

Group ID: 1 byte

TOS_Msg payload length = 1 byte (implies TOSH_DATA_LENGTH can be set to be max. 255) – does not include the CRC bytes or have anything to do with the serial/radio framing bytes!

TOS_Msg payload = (your app's data structure typically)

Or if Multihop routing is used by your app then,

a) TOS_Msg payload = MultihopMsg header (7 bytes) + (MHop Route Update, AM = 0xFA) and at other times,

b) TOS_Msg payload = MultihopMsg header (7 bytes) + (your app's data structure, max. TOSH_DATA_LENGTH - 7 bytes, AM = your app specific AM value)

where MultihopMsg header fields are:

Source: 2 bytes, address of the forwarding node (note this field will keep changing as the packet makes its way through the network!)

Origin: 2 bytes, address of the node that originated the packet (unlike the above field, does not change)

Sequence #: 2 bytes, used for determining route and calculating missed packets.

Hopcount: 1 byte, equals number of nodes traversed, used for calculating route

➤ If your app uses TinySec then frame payload format is as follows instead:

Frame payload = TinySec header (4 for TinySec-Auth mode or 8 bytes for TinySec-AE mode) + TinySec payload (max 29 bytes**) + MAC (4 bytes)

** unless TOSH_DATA_LENGTH is defined to be something different in your app. The default is defined in `tos/types/AM.h` and is set to a value of 29. Caution: Since the TinySec header's

payload-length field uses the upper two bits to encode the TinySec mode (TinySec-Auth or TinySec-AE) I suspect `TOSH_DATA_LENGTH` should only be set to 63 or lower value (when transmitting data using TinySec-Auth or TinySec-AE). If upper two bits are set to 10 (i.e. 0b10xxxxxx) mode is TinySec-Auth and if they are set to 11 (i.e. 0b11xx xxxx) then mode is TinySec-AE.

Consider for example that the TinySec mode is AE then we have TinySec header fields:

Destination: 2 bytes

AM type: 1 byte

TinySec payload length = 1 byte (length | 0b1100 0000 i.e. encoded with mode information as explained above) – does not include the MAC bytes or have anything to do with the radio frame bytes!

Counter = 2 bytes (starts from 0x00)

Source = 2 bytes

If instead the TinySec mode is Auth then we have TinySec header fields:

Destination: 2 bytes

AM type: 1 byte

TinySec payload length = 1 byte (length | 0b1000 0000 i.e. encoded with mode information as explained above) – does not include the MAC bytes or have anything to do with the radio frame bytes!

Of course the TinySec payload contents remain the same as before (non-TinySec case).

➤ *For TinySec enabled apps, once a frame reaches the base station, it seems the frame payload is converted to non-TinySec format (i.e. standard TinyOS `TOS_Msg` format, as plaintext not encrypted) and sent to the PC. In case the serial port method of transfer is being used (e.g. MIB510) the full frame comprises of:*

Frame = Frame begin sync byte (0x7E) + Frame type (1 byte) + Frame payload (always 36 bytes, if `TOSH_DATA_LENGTH` is 29 bytes) + Frame end sync byte (0x7E)

Since 0x7E is the frame sync code (to detect frames within the serial stream being received via the PC's com port), the frame payload uses 0x7D as an escape byte. Thus if any frame payload byte is 0x7E, it will be represented as 0x7D 0x5E instead.

Frame type	Frame code	Description
P_PACKET_NO_ACK	0x42	User packet with no ACK required.
P_PACKET_ACK	0x41	User packet with ACK required. Includes a prefix byte. Receiver must send a P_ACK response with prefix byte as contents.
P_ACK	0x40	The ACK response to a P_PACKET_ACK packet. Includes the prefix byte as its contents.
P_UNKNOWN	0xFF	Unknown packet type

Table 2: TinyOS, frame details

The above information is explained more with the below notes regarding the 1st test simulation and contains specific examples (when following the simulation output listings in the appendices please note that byte stream format is little endian!). The 2nd test's output should be self-explanatory.

- Transmission mode is switched to AE (Authenticate and Encrypt) right at the beginning (via `stdcontrol.init()` function in the TestM1A's *module* file)
- When multihop route update timer fires, the length of payload is initially 10 bytes (0x0A) (however gradually with time route update broadcast messages becomes larger as the routing tables form in the different nodes)
- Multihop 'route update' messages are broadcast i.e. 'destination' field of AE header is set to 0xFFFF with 'AM' field of AE header = 0xFA (i.e. decimal 250 for route update message)
- The AE header field 'length' includes encoding for the AE mode (upper two bits of length field byte i.e. 0b11xxxxxx)
- 4-byte MAC is computed and transmitted as the last 4 bytes of message

- 'Counter' field of AE header = 0x0000 (counter starts with 0)
- 'Source' field of AE header = 0x0001 (since mote 1 is transmitting)
- Entire header (i.e. AE mode IV) is sent in the clear (i.e. as plaintext). Next the contents of the payload (multihop header + either multihop route update or TestM1A application data i.e. `MistyMsg` structure) are encrypted and integrity protected using MAC mechanism (last 4 bytes).
- We notice the TestM1A application data being transmitted using Multihop routing is initially broadcast (since at this time the node's routing table has not formed properly). Much later we find the same is not broadcasted, instead transmitted specifically with destination = 0x<nodeID> in the AE header (where nodeID is the transmitting node's parent node)
- For TestM1A application data packets, we find total 29 bytes are radio transmitted by a non-basestation node (8 byte mode AE header + [7 byte + 10 byte `MistyMsg` structure] encrypted payload + 4 byte MAC)
- Node 0's radio receives entire message (MAC checks out OK)
- Node 0 sends extracted + decrypted + re-packaged message to the serial port (I used the Listen java tool with its source set to *tossim-serial* in order to listen to messages coming from tossim node 0). Also I use the term "re-packaged" because the TinySec AE header is stripped out and instead standard AM layer `TOS_Msg` header is used for serial transmission (5 byte `TOS_Msg` header + [7 byte multihop header + 10 byte `MistyMsg` structure] payload + 2 byte CRC-16). We find that the message received on *tossim-serial* has the following format (consult `TOS_Msg` structure explanation):
 - `TOS_Msg` header field 'Destination' = 0x007E (decimal 126 i.e. UART port). Note the multihop header is continuously changed if message were received by intermediary nodes in a multi-node simulation (i.e. source field = local address and destination field = parent address, at each parent node) during the multihop routing process. Node 0's parent is always UART (address 0x007E).
 - `TOS_Msg` header field 'AM' = 0x14 (i.e. decimal 20 for `AM_MSG1`)
 - `TOS_Msg` header field 'Group' = 0x7D (default group ID)
 - `TOS_Msg` header field 'Payload Length' = 0x11 (17 bytes = 7 byte multihop header + 10 byte `MistyMsg` structure)

- TOS_Msg payload bytes 1 & 2: multihop header, source field = 0x0000
- TOS_Msg payload bytes 3 & 4: multihop header, origin field = 0x0001 (when node 1 is sending the testdata)
- TOS_Msg payload bytes 5 & 6: multihop header, sequence # field = 0x004A
- TOS_Msg payload byte 7: multihop header, hopcount field = 0x0000 (from node 0 to UART)
- TOS_Msg payload byte 8: MistyMsg.testdata[0] = 0x01
- TOS_Msg payload byte 9: MistyMsg.testdata[1] = 0x23
- TOS_Msg payload byte 10: MistyMsg.testdata[2] = 0x45
- TOS_Msg payload byte 11: MistyMsg.testdata[3] = 0x67
- TOS_Msg payload byte 12: MistyMsg.testdata[4] = 0x89
- TOS_Msg payload byte 13: MistyMsg.testdata[5] = 0xab
- TOS_Msg payload byte 14: MistyMsg.testdata[6] = 0xcd
- TOS_Msg payload byte 15: MistyMsg.testdata[7] = 0xef
- TOS_Msg payload bytes 16 and 17: MistyMsg.datasourcesource = 0x0000 or 0x0001
- Note that 'Listen' does not display the following on screen:
 - Frame sync bytes at start and end of frame (0x7E)
 - Frame packet type (0x42 for P_Packet_NO_ACK)
 - CRC-16
- 'Listen' also "de-escapes" the incoming sequence i.e. converts each 0x7D 0x5E (multi-byte) to 0x7E. Note the message is transmitted as plaintext on the serial link. Example 'Listen' output:

7E 00 14 7D 11 00 00 01 00 4A 00 00 01 23 45 67 89 AB CD EF 01 00

where,

- 0x007E = TOS_Msg header field 'Destination' (UART address)
- 0x14 = TOS_Msg header field 'AM type' (AM_MSG1)
- 0x7D = TOS_Msg header field 'Group ID'
- 0x11 = TOS_Msg header field 'Payload Length' (17 bytes)
- 0x0000 = MultihopMsg header 'Packet Source'
- 0x0001 = MultihopMsg header 'Packet Origin'
- 0x004A = MultihopMsg header 'Sequence #'

- `0x00` = `MultihopMsg` header 'Hopcount' (from node x to base station, here node 0 to BS, hence 0 !)
- `0x0123456789abcdef` = TestM1A application data (`MistyMsg.testdata[0..7]`)
- `0x0001` = TestM1A application data (`MistyMsg.datasource`)

CHAPTER 6: Conclusion and Future Work

The Misty1 cipher was successfully implemented and integrated into existing TinySec framework. This means henceforth users of TinySec library have another option to choose from as regards the cipher (i.e. SkipJack, RC5 or Misty1). Software projects rely heavily on the concept of re-use in order to save programmers from “re-inventing the wheel”. This project demonstrates the same in two ways: (1) TinySec being modular in its architecture allowed easy replacement of the cipher component and indeed any suitable cipher can be implemented and integrated with it (2) A C-language implementation was picked up, ported to nesC language and modified to suit the project requirements. The testing results were positive showing that the implementation was correctly encrypting and decrypting data which was being provided to TinySec.

Code optimization is always a possibility for future work in any software project. A project extending this one could compare and contrast the critical parameters (performance, resource usage etc.) of the three ciphers mentioned in this report. One could also go for implementing AES (Rijndael cipher) into the TinySec library – this would most probably require modification and testing of other associated TinySec files such as CBCMAC.nc and CBCmodeM.nc etc (for example to support the 128-bit block cipher).

However, moving a step in a different direction, hardware based encryption is another way forward. Newer radio transceiver chips, such as the Texas Instruments Chipcon CC2420 (Single-Chip 2.4 GHz IEEE 802.15.4 Compliant and ZigBee(TM) Ready RF Transceiver) offer hardware based encryption features (e.g. AES-128). These chips are used for example in the newer MICAz motes (versus the earlier MICA and MICA2 motes) offered by Crossbow Technology Inc. There are some pointers available on the net written up by a student, Kun Sun, from NC State University and I figure additional/comprehensive information will become available once the hardware based approach becomes popular. This will help offload the encryption work presently being done by software onto hardware. If attempting this option, a student would have to be comfortable working with hardware specification/data sheets.

I plan to submit my contribution to the TinyOS group on the Internet and hope that it is included in the official TinyOS package in the future. In summary I'd like to say that I've gained substantially from this project in terms of topics such as wireless sensors networks, commercial WSN hardware (e.g. Motes manufactured by Crossbow) and related software (e.g. Moteworks, open-source TinyOS), cryptography/electronic-computer systems security, TOSSIM-TinyViz simulator package, TinySec library for TinyOS, Cygwin/Java/Makefiles/Programmer's Notepad (PN2), nesC programming, Mote hardware programming, porting of software etc.

APPENDIX A: Misty1M.nc, source code

```
// $Id: Misty1M.nc,v 1.0 2007/10/30 21:56:00 Ajay Mamtura Exp $

/*                                                    tab:4
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that this copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE TO ANY PARTY FOR DIRECT,
 * INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE AUTHOR
 * HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE AUTHOR SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE AUTHOR HAS NO OBLIGATION TO PROVIDE
 * MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 */

/* Authors: Ajay Mamtura
 * Date:    10/30/07
 */

/**
 * @author Ajay Mamtura
 */

/* Note: Original C source code version by Hironobu SUZUKI
 * as per details provided below. The original C source code
 * was ported, modified (to be compatible with current TinySec
 * library) and tested by Ajay Mamtura, Ryerson University, Toronto
 * email: amamtura@gmail.com
 *
 * Some useful/important usage details:
 * 1. In file crypto.h, define struct CipherContext as
 * "uint16_t context[32];" instead of "uint8_t context[20];"
 *
 * 2. Ensure wiring is changed in files: TinySecC.nc and TinySecAppC.nc
 * - In TinySecC.nc list the component "Misty1M as Cipher" instead
```

```

* of "SkipJackM as Cipher"
* - In TinySecAppC.nc list the component "MistylM" instead of "SKipJack"
* & wire "CBCModeM.BlockCipher -> MistylM.BlockCipher;"
*       "CBCModeM.BlockCipherInfo -> MistylM.BlockCipherInfo;"
* instead of "CBCModeM.BlockCipher -> SkipJackM.BlockCipher;"
* and       "CBCModeM.BlockCipherInfo -> SkipJackM.BlockCipherInfo;"
*/

```

```

/*
* Presented by H2NP.
*
* Name: MISTY1 module -- C source code version.
* See Also: Internet Draft --
*           A Description of the MISTY1 Encryption Algorithm
* Document File: draft-ohta-misty1desc-00.txt
*
* Author: Hironobu SUZUKI
* EMAIL: hironobu@h2np.suginami.tokyo.jp
* URL: http://www.pp.iij4u.or.jp/~h2np
* Copyright Notice: Copyright (C) 1998, Hironobu SUZUKI.
* Copyright Condition: GNU GENERAL PUBLIC LICENSE Version 2
* Date: 11 January 1998
* Date: 26 September 1999 ( 5% faster )
*
*/

```

```

#include crypto;

```

```

module MistylM {
    provides interface BlockCipher;
    provides interface BlockCipherInfo;
}

implementation {

    typedef struct M1Context {
        uint16_t skey[32];
    } M1Context;

    enum { BSIZE = 8 };

```

```
static const uint8_t s7[] =
```

```
{
```

```
0x1b, 0x32, 0x33, 0x5a, 0x3b, 0x10, 0x17, 0x54, 0x5b, 0x1a, 0x72, 0x73, 0x6b, 0x2c,  
0x66, 0x49, 0x1f, 0x24, 0x13, 0x6c, 0x37, 0x2e, 0x3f, 0x4a, 0x5d, 0x0f, 0x40, 0x56,  
0x25, 0x51, 0x1c, 0x04, 0x0b, 0x46, 0x20, 0x0d, 0x7b, 0x35, 0x44, 0x42, 0x2b, 0x1e,  
0x41, 0x14, 0x4b, 0x79, 0x15, 0x6f, 0x0e, 0x55, 0x09, 0x36, 0x74, 0x0c, 0x67, 0x53,  
0x28, 0x0a, 0x7e, 0x38, 0x02, 0x07, 0x60, 0x29, 0x19, 0x12, 0x65, 0x2f, 0x30, 0x39,  
0x08, 0x68, 0x5f, 0x78, 0x2a, 0x4c, 0x64, 0x45, 0x75, 0x3d, 0x59, 0x48, 0x03, 0x57,  
0x7c, 0x4f, 0x62, 0x3c, 0x1d, 0x21, 0x5e, 0x27, 0x6a, 0x70, 0x4d, 0x3a, 0x01, 0x6d,  
0x6e, 0x63, 0x18, 0x77, 0x23, 0x05, 0x26, 0x76, 0x00, 0x31, 0x2d, 0x7a, 0x7f, 0x61,  
0x50, 0x22, 0x11, 0x06, 0x47, 0x16, 0x52, 0x4e, 0x71, 0x3e, 0x69, 0x43, 0x34, 0x5c,  
0x58, 0x7d };
```

```
static const uint16_t s9[] =
```

```
{
```

```
0x1c3, 0x0cb, 0x153, 0x19f, 0x1e3, 0x0e9, 0x0fb, 0x035, 0x181, 0x0b9, 0x117, 0x1eb,  
0x133, 0x009, 0x02d, 0x0d3, 0x0c7, 0x14a, 0x037, 0x07e, 0x0eb, 0x164, 0x193, 0x1d8,  
0x0a3, 0x11e, 0x055, 0x02c, 0x01d, 0x1a2, 0x163, 0x118, 0x14b, 0x152, 0x1d2, 0x00f,  
0x02b, 0x030, 0x13a, 0x0e5, 0x111, 0x138, 0x18e, 0x063, 0x0e3, 0x0c8, 0x1f4, 0x01b,  
0x001, 0x09d, 0x0f8, 0x1a0, 0x16d, 0x1f3, 0x01c, 0x146, 0x07d, 0x0d1, 0x082, 0x1ea,  
0x183, 0x12d, 0x0f4, 0x19e, 0x1d3, 0x0dd, 0x1e2, 0x128, 0x1e0, 0x0ec, 0x059, 0x091,  
0x011, 0x12f, 0x026, 0x0dc, 0x0b0, 0x18c, 0x10f, 0x1f7, 0x0e7, 0x16c, 0x0b6, 0x0f9,  
0x0d8, 0x151, 0x101, 0x14c, 0x103, 0x0b8, 0x154, 0x12b, 0x1ae, 0x017, 0x071, 0x00c,  
0x047, 0x058, 0x07f, 0x1a4, 0x134, 0x129, 0x084, 0x15d, 0x19d, 0x1b2, 0x1a3, 0x048,  
0x07c, 0x051, 0x1ca, 0x023, 0x13d, 0x1a7, 0x165, 0x03b, 0x042, 0x0da, 0x192, 0x0ce,  
0x0c1, 0x06b, 0x09f, 0x1f1, 0x12c, 0x184, 0x0fa, 0x196, 0x1e1, 0x169, 0x17d, 0x031,  
0x180, 0x10a, 0x094, 0x1da, 0x186, 0x13e, 0x11c, 0x060, 0x175, 0x1cf, 0x067, 0x119,  
0x065, 0x068, 0x099, 0x150, 0x008, 0x007, 0x17c, 0x0b7, 0x024, 0x019, 0x0de, 0x127,  
0x0db, 0x0e4, 0x1a9, 0x052, 0x109, 0x090, 0x19c, 0x1c1, 0x028, 0x1b3, 0x135, 0x16a,  
0x176, 0x0df, 0x1e5, 0x188, 0x0c5, 0x16e, 0x1de, 0x1b1, 0x0c3, 0x1df, 0x036, 0x0ee,  
0x1ee, 0x0f0, 0x093, 0x049, 0x09a, 0x1b6, 0x069, 0x081, 0x125, 0x00b, 0x05e, 0x0b4,  
0x149, 0x1c7, 0x174, 0x03e, 0x13b, 0x1b7, 0x08e, 0x1c6, 0x0ae, 0x010, 0x095, 0x1ef,  
0x04e, 0x0f2, 0x1fd, 0x085, 0x0fd, 0x0f6, 0x0a0, 0x16f, 0x083, 0x08a, 0x156, 0x09b,  
0x13c, 0x107, 0x167, 0x098, 0x1d0, 0x1e9, 0x003, 0x1fe, 0x0bd, 0x122, 0x089, 0x0d2,  
0x18f, 0x012, 0x033, 0x06a, 0x142, 0x0ed, 0x170, 0x11b, 0x0e2, 0x14f, 0x158, 0x131,  
0x147, 0x05d, 0x113, 0x1cd, 0x079, 0x161, 0x1a5, 0x179, 0x09e, 0x1b4, 0x0cc, 0x022,  
0x132, 0x01a, 0x0e8, 0x004, 0x187, 0x1ed, 0x197, 0x039, 0x1bf, 0x1d7, 0x027, 0x18b,  
0x0c6, 0x09c, 0x0d0, 0x14e, 0x06c, 0x034, 0x1f2, 0x06e, 0x0ca, 0x025, 0x0ba, 0x191,  
0x0fe, 0x013, 0x106, 0x02f, 0x1ad, 0x172, 0x1db, 0x0c0, 0x10b, 0x1d6, 0x0f5, 0x1ec,  
0x10d, 0x076, 0x114, 0x1ab, 0x075, 0x10c, 0x1e4, 0x159, 0x054, 0x11f, 0x04b, 0x0c4,  
0x1be, 0x0f7, 0x029, 0x0a4, 0x00e, 0x1f0, 0x077, 0x04d, 0x17a, 0x086, 0x08b, 0x0b3,
```

```

0x171, 0x0bf, 0x10e, 0x104, 0x097, 0x15b, 0x160, 0x168, 0x0d7, 0x0bb, 0x066, 0x1ce,
0x0fc, 0x092, 0x1c5, 0x06f, 0x016, 0x04a, 0x0a1, 0x139, 0x0af, 0x0f1, 0x190, 0x00a,
0x1aa, 0x143, 0x17b, 0x056, 0x18d, 0x166, 0x0d4, 0x1fb, 0x14d, 0x194, 0x19a, 0x087,
0x1f8, 0x123, 0x0a7, 0x1b8, 0x141, 0x03c, 0x1f9, 0x140, 0x02a, 0x155, 0x11a, 0x1a1,
0x198, 0x0d5, 0x126, 0x1af, 0x061, 0x12e, 0x157, 0x1dc, 0x072, 0x18a, 0x0aa, 0x096,
0x115, 0x0ef, 0x045, 0x07b, 0x08d, 0x145, 0x053, 0x05f, 0x178, 0x0b2, 0x02e, 0x020,
0x1d5, 0x03f, 0x1c9, 0x1e7, 0x1ac, 0x044, 0x038, 0x014, 0x0b1, 0x16b, 0x0ab, 0x0b5,
0x05a, 0x182, 0x1c8, 0x1d4, 0x018, 0x177, 0x064, 0x0cf, 0x06d, 0x100, 0x199, 0x130,
0x15a, 0x005, 0x120, 0x1bb, 0x1bd, 0x0e0, 0x04f, 0x0d6, 0x13f, 0x1c4, 0x12a, 0x015,
0x006, 0x0ff, 0x19b, 0x0a6, 0x043, 0x088, 0x050, 0x15f, 0x1e8, 0x121, 0x073, 0x17e,
0x0bc, 0x0c2, 0x0c9, 0x173, 0x189, 0x1f5, 0x074, 0x1cc, 0x1e6, 0x1a8, 0x195, 0x01f,
0x041, 0x00d, 0x1ba, 0x032, 0x03d, 0x1d1, 0x080, 0x0a8, 0x057, 0x1b9, 0x162, 0x148,
0x0d9, 0x105, 0x062, 0x07a, 0x021, 0x1ff, 0x112, 0x108, 0x1c0, 0x0a9, 0x11d, 0x1b0,
0x1a6, 0x0cd, 0x0f3, 0x05c, 0x102, 0x05b, 0x1d9, 0x144, 0x1f6, 0x0ad, 0x0a5, 0x03a,
0x1cb, 0x136, 0x17f, 0x046, 0x0e1, 0x01e, 0x1dd, 0x0e6, 0x137, 0x1fa, 0x185, 0x08c,
0x08f, 0x040, 0x1b5, 0x0be, 0x078, 0x000, 0x0ac, 0x110, 0x15e, 0x124, 0x002, 0x1bc,
0x0a2, 0x0ea, 0x070, 0x1fc, 0x116, 0x15c, 0x04c, 0x1c2 };

```

```

inline uint16_t fi(uint16_t fi_in, uint16_t fi_key) {
    uint16_t d9, d7;

    d9 = s9[((fi_in >> 7) & 0x1ff)] ^ (fi_in & 0x7f);
    d7 = (fi_in & 0x7f);
    d7 = ((s7[d7] ^ d9) & 0x7f) ^ ((fi_key >> 9) & 0x7f);
    d9 ^= (fi_key & 0x1ff);
    d9 = s9[d9] ^ d7;
    return ((d7<<9) | d9);
}

```

```

inline uint32_t fo(uint16_t *skey, uint32_t fo_in, uint8_t k) {
    uint16_t t0, t1;

    t0 = fi((skey[k]^((fo_in >> 16))), skey[((k+5)%8) + 8])^(fo_in & 0xFFFF);
    t1 = fi((fo_in & 0xFFFF)^skey[(k+2)%8], skey[((k+1)%8) + 8]);
    t1 ^= t0;
    t0 ^= skey[(k+7)%8];
    t0 = fi(t0, skey[((k+3)%8)+8])^t1;
    t1 ^= skey[(k+4)%8];
    return ( (t1<<16) | t0 );
}

```



```

inline uint32_t fl(uint16_t *skey, uint32_t fl_in, uint8_t k) {
    uint16_t d0, d1;
    uint8_t t;

    if ( k % 2 ) {
        t=(k-1)/2;
        d1 = (fl_in & 0xffff) ^ ((fl_in >> 16) & skey[((t+2)%8)+8]);
        d0 = (fl_in >> 16) ^ (d1 | skey[(t+4)%8]);
    }
    else {
        t=k/2;
        d1 = (fl_in & 0xffff) ^ ((fl_in >> 16) & skey[t]);
        d0 = (fl_in >> 16) ^ (d1 | skey[((t+6)%8)+8]);
    }

    return ((d0<<16) | d1);
}

inline uint32_t flinv(uint16_t *skey, uint32_t fl_in, uint8_t k) {
    uint16_t d0, d1;
    uint8_t t;

    d0 = (fl_in >> 16);
    d1 = fl_in & 0xffff;

    if ( k % 2 ) {
        t= (k-1)/2;
        d0 = d0 ^ (d1 | skey[(t+4)%8]);
        d1 = d1 ^ (d0 & skey[((t+2)%8)+8]);
    }
    else {
        t = k/2;
        d0 = d0 ^ (d1 | skey[((t+6)%8)+8]);
        d1 = d1 ^ (d0 & skey[t]);
    }

    return ((d0<<16) | d1);
}

```

```

/**
 * Encrypts a single block (of blockSize) using the key in the keySize.
 *
 * @param context holds the module specific opaque data related to the
 *         key (perhaps key expansions).
 * @param plainBlock a plaintext block of blockSize
 * @param cipherBlock the resulting ciphertext block of blockSize
 *
 * @return Whether the encryption was successful. Possible failure reasons
 *         include not calling init().
 */
async command result_t BlockCipher.encrypt(CipherContext *context, uint8_t *plainBlock,
uint8_t *cipherBlock)
{
    register uint16_t *skey = ((M1Context *)context->context)->skey; // ???
    register uint16_t w1, w2, w3, w4;
    register uint32_t d0, d1;

    c2sM(plainBlock, w1);
    plainBlock += 2;
    c2sM(plainBlock, w2);
    plainBlock += 2;
    c2sM(plainBlock, w3);
    plainBlock += 2;
    c2sM(plainBlock, w4);
    plainBlock += 2;

    d0 = (w1 << 16) | w2;
    d1 = (w3 << 16) | w4;

    /* 0 round */
    d0 = f1(skey,d0,0);
    d1 = f1(skey,d1,1);
    d1 = d1 ^ fo(skey,d0,0);

    /* 1 */
    d0 = d0 ^ fo(skey,d1,1);

    /* 2 */
    d0 = f1(skey,d0,2);
    d1 = f1(skey,d1,3);
    d1 = d1 ^ fo(skey,d0,2);

```

```

/* 3 */
    d0 = d0 ^ fo(skey,d1,3);

/* 4 */
    d0 = fl(skey,d0,4);
    d1 = fl(skey,d1,5);
    d1 = d1 ^ fo(skey,d0,4);

/* 5 */
    d0 = d0 ^ fo(skey,d1,5);

/* 6 */
    d0 = fl(skey,d0,6);
    d1 = fl(skey,d1,7);
    d1 = d1 ^ fo(skey,d0,6);

/* 7 */
    d0 = d0 ^ fo(skey,d1,7);

/* 8 */
    d0 = fl(skey,d0,8);
    d1 = fl(skey,d1,9);

/* round finished */
    w4 = (uint16_t)( d0 & 0x0000FFFF );
    w3 = (uint16_t)( (d0 & 0xFFFF0000) >> 16 );
    w2 = (uint16_t)( d1 & 0x0000FFFF );
    w1 = (uint16_t)( (d1 & 0xFFFF0000) >> 16);

    s2cM(w1,cipherBlock);
    cipherBlock += 2;
    s2cM(w2,cipherBlock);
    cipherBlock += 2;
    s2cM(w3,cipherBlock);
    cipherBlock += 2;
    s2cM(w4,cipherBlock);
    cipherBlock += 2;

    return SUCCESS;
}

```

```

/**
 * Decrypts a single block (of blockSize) using the key in the keySize. Not
 * all ciphers will implement this function (since providing encryption
 * is a useful primitive).
 *
 * @param context holds the module specific opaque data related to the
 *       key (perhaps key expansions).
 * @param cipherBlock a ciphertext block of blockSize
 * @param plainBlock the resulting plaintext block of blockSize
 *
 * @return Whether the decryption was successful. Possible failure reasons
 *       include not calling init() or an unimplimented decrypt function.
 */
async command result_t BlockCipher.decrypt(CipherContext *context, uint8_t
*cipherBlock, uint8_t *plainBlock)
{
    register uint16_t *skey = ((M1Context *)context->context)->skey; // ???
    register uint16_t w1, w2, w3, w4;
    register uint32_t d0, d1;

    c2sM(cipherBlock, w1);
    cipherBlock += 2;
    c2sM(cipherBlock, w2);
    cipherBlock += 2;
    c2sM(cipherBlock, w3);
    cipherBlock += 2;
    c2sM(cipherBlock, w4);

    d0 = (w3 << 16) | w4;
    d1 = (w1 << 16) | w2;

    d0 = flinv(skey,d0, 8);
    d1 = flinv(skey,d1, 9);
    d0 = d0 ^ fo(skey,d1, 7);
    d1 = d1 ^ fo(skey,d0, 6);
    d0 = flinv(skey,d0, 6);
    d1 = flinv(skey,d1, 7);
    d0 = d0 ^ fo(skey,d1, 5);
    d1 = d1 ^ fo(skey,d0, 4);
    d0 = flinv(skey,d0, 4);
    d1 = flinv(skey,d1, 5);
    d0 = d0 ^ fo(skey,d1, 3);

```

```

d1 = d1 ^ fo(skey,d0, 2);
d0 = flinv(skey,d0, 2);
d1 = flinv(skey,d1, 3);
d0 = d0 ^ fo(skey,d1, 1);
d1 = d1 ^ fo(skey,d0, 0);
d0 = flinv(skey,d0, 0);
d1 = flinv(skey,d1, 1);

w4 = (uint16_t)( d1 & 0x0000FFFF );
w3 = (uint16_t)( (d1 & 0xFFFF0000) >> 16 );
w2 = (uint16_t)( d0 & 0x0000FFFF );
w1 = (uint16_t)( (d0 & 0xFFFF0000) >> 16);

s2cM(w1,plainBlock);
plainBlock += 2;
s2cM(w2,plainBlock);
plainBlock += 2;
s2cM(w3,plainBlock);
plainBlock += 2;
s2cM(w4,plainBlock);

return SUCCESS;
}

/**
 * Performs the key extension and expansion on the real secret.
 *
 * @param secret key
 */
result_t setupKey (CipherContext *context, uint8_t *key, uint8_t keySize) {
    int i;
    uint8_t k[16];
    uint16_t ek[32];
    uint16_t *skey = ((M1Context *)context->context)->skey;
    // casting ptr to 32x2-byte array, shouldn't there be
    // '&' operator right at begining of LHS ???

    // (NOTE: this is NOT part of the Mistyl algo!)
    // creating a local extended-copy of cipher key (128-bit) by
    // duplicating and concatenating the original 64-bit Encryption Key

```

```

for (i=0; i < 8; i++) {
    k[i] = *(key+i);    // k[0]..k[7]
}
for (i=0; i < 8; i++) {
    k[i+8] = *(key+i);  // k[8]..k[15]
}
/* Begin debug key
    k[0] = 0x00;
    k[1] = 0x11;
    k[2] = 0x22;
    k[3] = 0x33;
    k[4] = 0x44;
    k[5] = 0x55;
    k[6] = 0x66;
    k[7] = 0x77;
    k[8] = 0x88;
    k[9] = 0x99;
    k[10] = 0xaa;
    k[11] = 0xbb;
    k[12] = 0xcc;
    k[13] = 0xdd;
    k[14] = 0xee;
    k[15] = 0xff;
End debug key */
for (i=0; i < 16; i++) {
    dbg(DBG_CRYPTO, "k[%u] = %x\n", i, k[i]);
}

// NOTE: NOW this is required as per Misty1 algo!
// We create expanded key (sub-key???) (32 x 2-byte) based on the Misty1 cipher algo
for(i=0; i < 8; i++) {
    ek[i] = (k[i*2]*256) + (k[(i*2)+1]);
}

for (i=0 ; i < 8; i++) {
    ek[i+8] = fi(ek[i], ek[(i+1)%8]);
    ek[i+16] = ek[i+8] & 0x1ff;
    ek[i+24] = ek[i+8] >> 9;
}
skey = &ek[0];
return SUCCESS;
}

```

```

/**
 * Initialize the BlockCipher context.
 *
 * @param context structure to hold the opaque data from this initialization
 *        call. It should be passed to future invocations of this module
 *        which use this particular key.
 * @param blockSize size of the block in bytes. Some cipher implementation
 *        may support multiple block sizes, in which case any valid size
 *        is valid.
 * @param keySize key size in bytes
 * @param key pointer to the key
 *
 * @return Whether initialization was successful. The command may be
 *        unsuccessful if the key size or blockSize are not valid for the
 *        given cipher implementation.
 */
command result_t BlockCipher.init(CipherContext *context, uint8_t blockSize, uint8_t
keySize, uint8_t *key) {
    if (blockSize != BSIZE) {
        return FAIL;
    }
    dbg(DBG_USR3, "Cipher: Misty1\n");
    return setupKey (context, key, keySize);
}

/**
 * Returns the preferred block size that this cipher operates with. It is
 * always safe to call this function before the init() call has been made.
 *
 * @return the preferred block size for this cipher. In the case where the
 *        cipher operates with multiple block sizes, this will pick one
 *        particular size (deterministically).
 */
async command uint8_t BlockCipherInfo.getPreferredBlockSize() {
    return BSIZE;
}
}

```

APPENDIX B: crypto.h, source code

```
// $Id: crypto.h,v 1.3 2004/05/18 19:16:33 nksrules Exp $

/*                                     tab:4
 * "Copyright (c) 2000-2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704. Attention: Intel License Inquiry.
 */

/* Authors: Naveen Sastry
 * Date: 10/20/02
 *
 * Basic header file descriptions for the crpto contexts. These are untyped
 * and meant to be opaque to all but the appropriate modules, which should
 * cast the structures to their appropriate internal types.
 */
```



```

/*
 * Context for block cipher.
 */

/**
 * @author Naveen Sastry
 */

typedef struct CipherContext {
    // rc5      needs 104 bytes
    // skipjack needs 20 bytes
    // mistyl   needs 64 bytes
    uint16_t context[32];
} CipherContext;

/**
 * Context for the MAC
 */
typedef struct MACContext {
    CipherContext cc;
    uint8_t context[12];
} MACContext;

/**
 * Context for the block cipher modes
 */
typedef struct CipherModeContext {
    CipherContext cc;
    uint8_t context[24];
} CipherModeContext;

```

APPENDIX C.1: Makefile (Test1, nesC compiler), source code

```
PFLAGS= -I%T/lib/Route -I%T/lib/Queue -I%T/lib/Broadcast
COMPONENT=TestM1A
#TINYSEC=true
GOALS += tinysec
include ../Makerules
```

APPENDIX C.2: TestM1A.h (Test1, nesC header file), source code

```
typedef struct MistyMsg {
    uint8_t testdata[8];
    uint16_t datasource;
    // uint16_t parentaddr;
} __attribute__((packed)) MistyMsg;

enum {
    AM_MSG1 = 20,
    AM_MSG2 = 40
};
```

APPENDIX C.3: TestM1A.nc (Test1, *configuration* file), source code

```
includes TestM1A;
includes MultiHop;

configuration TestM1A {
}

implementation {
    components Main, TestM1AM, TimerC, LedsC, GenericCommPromiscuous as Comm,
        Bcast, MultiHopRouter as multihopM, QueuedSend, TinySecC;

    Main.StdControl -> TestM1AM.StdControl;
    Main.StdControl -> Bcast.StdControl;
    Main.StdControl -> multihopM.StdControl;
    Main.StdControl -> QueuedSend.StdControl;
    Main.StdControl -> TimerC.StdControl;
    Main.StdControl -> Comm;
    // multihopM.CommControl -> Comm;

    TestM1AM.Timer -> TimerC.Timer[unique("Timer")];
    TestM1AM.Leds -> LedsC;

    TestM1AM.Bcast -> Bcast.Receive[AM_MSG2];
    Bcast.ReceiveMsg[AM_MSG2] -> Comm.ReceiveMsg[AM_MSG2];

    TestM1AM.RouteControl -> multihopM;
    TestM1AM.Send -> multihopM.Send[AM_MSG1];
    multihopM.ReceiveMsg[AM_MSG1] -> Comm.ReceiveMsg[AM_MSG1];
    //multihopM.ReceiveMsg[AM_MULTIHOPMSG] -> Comm.ReceiveMsg[AM_MULTIHOPMSG];

    TestM1AM.TinySecMode -> TinySecC.TinySecMode;
}
```

APPENDIX C.4: TestM1AM.nc (Test1, *module* file), source code

```
includes TestM1A;

module TestM1AM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
    interface Send;
    interface Receive as Bcast;
    interface RouteControl;
    interface TinySecMode;
  }
}

implementation {

  TOS_Msg gMsgBuffer;

  /*****
   * Initialization
   *****/

  task void SendData() {
    MistyMsg *pTest;
    uint16_t Len;
    dbg(DBG_USR3, "TestM1AM: Sending testdata\n");

    if ( (pTest = (MistyMsg *)call Send.getBuffer(&gMsgBuffer,&Len)) ) {
      pTest->testdata[0] = 0x01;
      pTest->testdata[1] = 0x23;
      pTest->testdata[2] = 0x45;
      pTest->testdata[3] = 0x67;
      pTest->testdata[4] = 0x89;
      pTest->testdata[5] = 0xab;
      pTest->testdata[6] = 0xcd;
      pTest->testdata[7] = 0xef;
      pTest->datasource = TOS_LOCAL_ADDRESS;
      call Send.send(&gMsgBuffer,sizeof(MistyMsg));
    }
  }
}
```

```

command result_t StdControl.init() {
    call Leds.init();
    call TinySecMode.setTransmitMode(TINYSEC_ENCRYPT_AND_AUTH);
    switch(call TinySecMode.getTransmitMode()) {
        case 1:
            dbg(DBG_USR3, "TestM1AM: TinySec transmit mode is AUTH_ONLY\n");
            break;
        case 2:
            dbg(DBG_USR3, "TestM1AM: TinySec transmit mode is ENCRYPT_AND_AUTH\n");
            break;
        case 3:
            dbg(DBG_USR3, "TestM1AM: TinySec transmit mode is DISABLED (No TinySec)\n");
            break;
        default:
            dbg(DBG_USR3, "TestM1AM: TinySec transmit mode is ? (Invalid)\n");
            break;
    }
    return SUCCESS;
}

command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 5000);
}

command result_t StdControl.stop() {
    return call Timer.stop();
}

/*****
* Commands and events
*****/
event result_t Timer.fired() {
    dbg(DBG_USR3, "TestM1AM Timer fired\n");
    post SendData();
    return SUCCESS;
}

event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) {
    dbg(DBG_USR3, "TestM1AM: Sending testdata complete 0x%x\n", success);
    return SUCCESS;
}

```

```
/* Command interpreter for broadcasts
 *
 */
event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload, uint16_t payloadLen)
{
    return pMsg;
}

}
```

APPENDIX D.1: Makefile (Test2, nesC compiler), source code

```
COMPONENT=TestM1B
#TINYSEC=true
GOALS += tinysec
include ../Makerules
```

APPENDIX D.2: TestM1B.nc (Test2, *configuration* file), source code

```
configuration TestM1B {
}
implementation {
  components Main, TestM1BM, TimerC, LedsC, TinySecAppC;

  Main.StdControl -> TestM1BM.StdControl;
  Main.StdControl -> TimerC.StdControl;

  TestM1BM.Timer -> TimerC.Timer[unique("Timer")];
  TestM1BM.Leds -> LedsC;

  TestM1BM.TinySecApp -> TinySecAppC.TinySecApp;
}
```

APPENDIX D.3: TestM1B.nc (Test2, *module* file), source code

```
module TestM1BM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
    interface TinySecApp;
  }
}

implementation {

  // Note: if "debug key" code is enabled in Misty1M.nc, below key does not matter !
  uint8_t apptestkey[] = {0x01,0x23,0x45,0x67,0x89,0xAB,0xCD,0xEF};
  bool glbkey = FALSE;
  uint8_t srcptext[] = {0x54,0x45,0x53,0x54,0x49,0x4E,0x47,0x21}; // TESTING!
  uint8_t ptlen = 8;
  uint8_t ctlen = 8;
  uint8_t ivlen = 8;
  uint8_t iv[] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
  uint8_t rcvrptext[] = {0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41}; // A
  uint8_t ctext[] = {0x5A,0x5A,0x5A,0x5A,0x5A,0x5A,0x5A,0x5A}; // Z

  /*****
   * Initialization
   *****/

  task void EncData() {
    dbg(DBG_USR3, "TestM1BM: Proceeding to encrypt data\n");
    call TinySecApp.encryptData(&srcptext[0], ptlen, ivlen, &iv[0], &ctext[0]);
  }

  task void DecData() {
    dbg(DBG_USR3, "TestM1BM: Proceeding to decrypt data\n");
    call TinySecApp.decryptData(&ctext[0], ctlen, ivlen, &iv[0], &rcvrptext[0]);
  }

  task void printStart() {
    int i;
    dbg(DBG_USR3, "TestM1BM: APP KEY = ");
    for (i=0; i<8; i++)
```



```

        dbg(DBG_USR3, "0x%x ", apptestkey[i]);
    dbg(DBG_USR3, "\n");

    dbg(DBG_USR3, "TestM1BM: SOURCE PLAINTEXT = ");
    for (i=0; i<8; i++)
        dbg(DBG_USR3, "%c ", srcptext[i]);
    dbg(DBG_USR3, "\n");

    dbg(DBG_USR3, "TestM1BM: IV = ");
    for (i=0; i<8; i++)
        dbg(DBG_USR3, "0x%x ", iv[i]);
    dbg(DBG_USR3, "\n");

    dbg(DBG_USR3, "TestM1BM: CIPHERTEXT = ");
    for (i=0; i<8; i++)
        dbg(DBG_USR3, "%c ", ctext[i]);
    dbg(DBG_USR3, "\n");

    dbg(DBG_USR3, "TestM1BM: RECOVERED P-TEXT (pre-enc) = ");
    for (i=0; i<8; i++)
        dbg(DBG_USR3, "%c ", rcvrptext[i]);
    dbg(DBG_USR3, "\n");
}

task void printEnd() {
    int i;
    dbg(DBG_USR3, "TestM1BM: APP KEY = ");
    for (i=0; i<8; i++)
        dbg(DBG_USR3, "0x%x ", apptestkey[i]);
    dbg(DBG_USR3, "\n");

    dbg(DBG_USR3, "TestM1BM: SOURCE PLAINTEXT = ");
    for (i=0; i<8; i++)
        dbg(DBG_USR3, "%c ", srcptext[i]);
    dbg(DBG_USR3, "\n");

    dbg(DBG_USR3, "TestM1BM: IV = ");
    for (i=0; i<8; i++)
        dbg(DBG_USR3, "0x%x ", iv[i]);
    dbg(DBG_USR3, "\n");

    dbg(DBG_USR3, "TestM1BM: CIPHERTEXT = ");

```

```

    for (i=0; i<8; i++)
        dbg(DBG_USR3, "%c ", ctext[i]);
    dbg(DBG_USR3, "\n");

    dbg(DBG_USR3, "TestM1BM: RECOVERED P-TEXT (post-dec) = ");
    for (i=0; i<8; i++)
        dbg(DBG_USR3, "%c ", rcvrptext[i]);
    dbg(DBG_USR3, "\n");

    memset(rcvrptext, 0x41, 8); // reset to A for next timer round
    memset(ctext, 0x5A, 8); // reset to Z for next timer round
}

command result_t StdControl.init() {
    call Leds.init();
    call TinySecApp.init(&apptestkey[0], glbkey);
    return SUCCESS;
}

command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 5000);
}

command result_t StdControl.stop() {
    return call Timer.stop();
}

/*****
 * Commands and events
 *****/
event result_t Timer.fired() {
    dbg(DBG_USR3, "TestM1BM: Timer fired\n");
    post printStart();
    post EncData();
    return SUCCESS;
}

event result_t TinySecApp.encryptDataDone(result_t result, uint8_t* ciphertext) {
    int i;
    if (result == SUCCESS) {
        dbg(DBG_USR3, "TestM1BM: Data encryption done\n");
    }
}

```

```

        dbg(DBG_USR3, "TestM1BM: CIPHERTEXT (post-enc) = ");
        for (i=0; i<8; i++)
            dbg(DBG_USR3, "%c ", ctext[i]);
        dbg(DBG_USR3, "\n");

        post DecData();
        return SUCCESS;
    }
    else
        return FAIL;
}

event result_t TinySecApp.decryptDataDone(result_t result, uint8_t* plaintext) {
    if (result == SUCCESS) {
        dbg(DBG_USR3, "TestM1BM: Data decryption done\n");
        post printEnd();
        return SUCCESS;
    }
    else
        return FAIL;
}
}

```

APPENDIX E.1: Test1 TOSSIM/Simulation Output

(partial listing only; note full-listing = 205 pages)

```
SIM: Random seed is 62800
SIM: Time for mote 0 initialized to 19176051.
SIM: Time for mote 1 initialized to 13881933.
1: TestM1M: TinySec transmit mode is ENCRYPT_AND_AUTH
1: MultiHopLEPSM timer task.
1: addr prnt misd rcvd lstS hop rEst sEst
1: MultiHopLEPSM Sending route update msg.
1: QueuedSend: queue msg enq 0 deq 0
1: QueuedSend: Successfully queued msg to 0xffff, enq 1, deq 0
1: QueuedSend: sending msg (0x0)
1: TINYSEC: sendInit() Sending length = 10
1: TINYSEC: sendInit() Length field = 202
1: bc: 0
1: CBC.encrypt j: 2; bc: 0
1: MAC: computed: a7 f 16 c0
1: TINYSEC: getTransmitByte() byte=1 data=ff
1: TINYSEC: getTransmitByte() byte=2 data=ff
1: TINYSEC: getTransmitByte() byte=3 data=fa
1: TINYSEC: getTransmitByte() byte=4 data=ca
1: TINYSEC: getTransmitByte() byte=5 data=0
1: TINYSEC: getTransmitByte() byte=6 data=0
1: TINYSEC: getTransmitByte() byte=7 data=1
1: TINYSEC: getTransmitByte() byte=8 data=0
1: TINYSEC: getTransmitByte() byte=9 data=1b
1: TINYSEC: getTransmitByte() byte=10 data=60
1: TINYSEC: getTransmitByte() byte=11 data=9c
1: TINYSEC: getTransmitByte() byte=12 data=c4
1: TINYSEC: getTransmitByte() byte=13 data=48
1: TINYSEC: getTransmitByte() byte=14 data=80
1: TINYSEC: getTransmitByte() byte=15 data=1a
1: TINYSEC: getTransmitByte() byte=16 data=38
1: TINYSEC: getTransmitByte() byte=17 data=40
1: TINYSEC: getTransmitByte() byte=18 data=2a
1: TINYSEC: getTransmitByte() byte=19 data=a7
1: TINYSEC: getTransmitByte() byte=20 data=f
1: TINYSEC: getTransmitByte() byte=21 data=16
1: TINYSEC: getTransmitByte() byte=22 data=c0
1: TINYSEC: getTransmitByte() signaling send done
1: MicaHighSpeedRadioTinySecM: Transmit done.
```

```

1: Rcvd ACK 0
1: qent 0 dequeued.
0: TestM1M: TinySec transmit mode is ENCRYPT_AND_AUTH
0: MultiHopLEPSM timer task.
0: addr prnt misd rcvd lstS hop rEst sEst
0: MultiHopLEPSM: Parent = 126
0: MultiHopLEPSM Sending route update msg.
0: QueuedSend: queue msg enq 0 deq 0
0: QueuedSend: Successfully queued msg to 0xffff, enq 1, deq 0
0: QueuedSend: sending msg (0x0)
0: TINYSEC: sendInit() Sending length = 10
0: TINYSEC: sendInit() Length field = 202
0: bc: 0
0: CBC.encrypt j: 2; bc: 0
0: MAC: computed: cb c4 6c d2
0: TINYSEC: getTransmitByte() byte=1 data=ff
0: TINYSEC: getTransmitByte() byte=2 data=ff
0: TINYSEC: getTransmitByte() byte=3 data=fa
1: Header received byte 1
1: TINYSEC: byteReceived() RxByteCnt=1 data=ff recDataLength=0 macRecCount=1
decryptRecCount=-7.
0: TINYSEC: getTransmitByte() byte=4 data=ca
1: Header received byte 2
1: TINYSEC: byteReceived() RxByteCnt=2 data=ff recDataLength=0 macRecCount=2
decryptRecCount=-6.
0: TINYSEC: getTransmitByte() byte=5 data=0
1: Header received byte 3
1: TINYSEC: byteReceived() RxByteCnt=3 data=fa recDataLength=0 macRecCount=3
decryptRecCount=-5.
0: TINYSEC: getTransmitByte() byte=6 data=0
1: Header received byte 4
1: TINYSEC: byteReceived() RxByteCnt=4 data=ca recDataLength=0 macRecCount=4
decryptRecCount=-4.
1: Detected TinySec bit.
1: Encryption enabled.
1: Receive init done TinySec.
1: MAC init called: rxlength=18.
0: TINYSEC: getTransmitByte() byte=7 data=0
1: TINYSEC: byteReceived() RxByteCnt=5 data=0 recDataLength=10 macRecCount=5
decryptRecCount=-3.
0: TINYSEC: getTransmitByte() byte=8 data=0
1: TINYSEC: byteReceived() RxByteCnt=6 data=0 recDataLength=10 macRecCount=6

```

```

decryptRecCount=-2.
0: TINYSEC: getTransmitByte() byte=9 data=4f
1: TINYSEC: byteReceived() RxByteCnt=7 data=0 recDataLength=10 macRecCount=7
decryptRecCount=-1.
0: TINYSEC: getTransmitByte() byte=10 data=5d
1: TINYSEC: byteReceived() RxByteCnt=8 data=0 recDataLength=10 macRecCount=0
decryptRecCount=0.
1: DECRYPT: init size=10
0: TINYSEC: getTransmitByte() byte=11 data=ad
1: TINYSEC: byteReceived() RxByteCnt=9 data=4f recDataLength=10 macRecCount=1
decryptRecCount=1.
0: TINYSEC: getTransmitByte() byte=12 data=f6
1: TINYSEC: byteReceived() RxByteCnt=10 data=5d recDataLength=10 macRecCount=2
decryptRecCount=2.
0: TINYSEC: getTransmitByte() byte=13 data=9d
1: TINYSEC: byteReceived() RxByteCnt=11 data=ad recDataLength=10 macRecCount=3
decryptRecCount=3.
0: TINYSEC: getTransmitByte() byte=14 data=ef
1: TINYSEC: byteReceived() RxByteCnt=12 data=f6 recDataLength=10 macRecCount=4
decryptRecCount=4.
0: TINYSEC: getTransmitByte() byte=15 data=f3
1: TINYSEC: byteReceived() RxByteCnt=13 data=9d recDataLength=10 macRecCount=5
decryptRecCount=5.
0: TINYSEC: getTransmitByte() byte=16 data=78
1: TINYSEC: byteReceived() RxByteCnt=14 data=ef recDataLength=10 macRecCount=6
decryptRecCount=6.
0: TINYSEC: getTransmitByte() byte=17 data=3c
1: TINYSEC: byteReceived() RxByteCnt=15 data=f3 recDataLength=10 macRecCount=7
decryptRecCount=7.
0: TINYSEC: getTransmitByte() byte=18 data=86
1: TINYSEC: byteReceived() RxByteCnt=16 data=78 recDataLength=10 macRecCount=0
decryptRecCount=8.
1: CBCModeM:incrementalDecrypt: <entry>length 8
1: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
1: CBCModeM: incrementalDecrypt. State 2LEFTA
1: DBCModeM: ** Switched to state 2LEFTB with 2 remaining
0: TINYSEC: getTransmitByte() byte=19 data=cb
1: TINYSEC: byteReceived() RxByteCnt=17 data=3c recDataLength=10 macRecCount=1
decryptRecCount=9.
0: TINYSEC: getTransmitByte() byte=20 data=c4
1: TINYSEC: byteReceived() RxByteCnt=18 data=86 recDataLength=10 macRecCount=2
decryptRecCount=10.

```

```

1: CBCModeM:incrementalDecrypt: <entry>length 2
1: CBCModeM: incrementalDecrypt. State 2LEFTB
1: incrementalDecrypt: 0 2
1: incrementalDecrypt: 1 2
0: TINYSEC: getTransmitByte() byte=21 data=6c
1: TINYSEC: byteReceived() RxByteCnt=19 data=cb recDataLength=10 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: getTransmitByte() byte=22 data=d2
0: TINYSEC: getTransmitByte() signaling send done
0: MicaHighSpeedRadioTinySecM: Transmit done.
1: TINYSEC: byteReceived() RxByteCnt=20 data=c4 recDataLength=10 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: byteReceived() RxByteCnt=21 data=6c recDataLength=10 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: byteReceived() RxByteCnt=22 data=d2 recDataLength=10 macRecCount=-1
decryptRecCount=-1.
1: MAC computed: cb c4 6c d2, MAC received: cb c4 6c d2
0: Rcvd ACK 1
0: gent 0 dequeued.
1: TestMIM Timer fired
1: TestMIM: Sending testdata
1: QueuedSend: queue msg enq 1 deq 1
1: QueuedSend: Successfully queued msg to 0xffff, enq 2, deq 1
1: QueuedSend: sending msg (0x1)
1: TINYSEC: sendInit() Sending length = 17
1: TINYSEC: sendInit() Length field = 209
1: bc: 8
1: CBC.encrypt j: 1; bc: 8
1: MAC: computed: c2 b4 45 62
1: TINYSEC: getTransmitByte() byte=1 data=ff
1: TINYSEC: getTransmitByte() byte=2 data=ff
1: TINYSEC: getTransmitByte() byte=3 data=14
0: Header received byte 1
0: TINYSEC: byteReceived() RxByteCnt=1 data=ff recDataLength=0 macRecCount=1
decryptRecCount=-7.
1: TINYSEC: getTransmitByte() byte=4 data=d1
0: Header received byte 2
0: TINYSEC: byteReceived() RxByteCnt=2 data=ff recDataLength=0 macRecCount=2
decryptRecCount=-6.
1: TINYSEC: getTransmitByte() byte=5 data=1
0: Header received byte 3
0: TINYSEC: byteReceived() RxByteCnt=3 data=14 recDataLength=0 macRecCount=3

```

```

decryptRecCount=-5.
1: TINYSEC: getTransmitByte() byte=6 data=0
0: Header received byte 4
0: TINYSEC: byteReceived() RxByteCnt=4 data=d1 recDataLength=0 macRecCount=4
decryptRecCount=-4.
0: Detected TinySec bit.
0: Encryption enabled.
0: Receive init done TinySec.
0: MAC init called: rxlength=25.
1: TINYSEC: getTransmitByte() byte=7 data=1
0: TINYSEC: byteReceived() RxByteCnt=5 data=1 recDataLength=17 macRecCount=5
decryptRecCount=-3.
1: TINYSEC: getTransmitByte() byte=8 data=0
0: TINYSEC: byteReceived() RxByteCnt=6 data=0 recDataLength=17 macRecCount=6
decryptRecCount=-2.
1: TINYSEC: getTransmitByte() byte=9 data=e3
0: TINYSEC: byteReceived() RxByteCnt=7 data=1 recDataLength=17 macRecCount=7
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=10 data=3
0: TINYSEC: byteReceived() RxByteCnt=8 data=0 recDataLength=17 macRecCount=0
decryptRecCount=0.
0: DECRYPT: init size=17
1: TINYSEC: getTransmitByte() byte=11 data=1c
0: TINYSEC: byteReceived() RxByteCnt=9 data=e3 recDataLength=17 macRecCount=1
decryptRecCount=1.
1: TINYSEC: getTransmitByte() byte=12 data=5c
0: TINYSEC: byteReceived() RxByteCnt=10 data=3 recDataLength=17 macRecCount=2
decryptRecCount=2.
1: TINYSEC: getTransmitByte() byte=13 data=d6
0: TINYSEC: byteReceived() RxByteCnt=11 data=1c recDataLength=17 macRecCount=3
decryptRecCount=3.
1: TINYSEC: getTransmitByte() byte=14 data=db
0: TINYSEC: byteReceived() RxByteCnt=12 data=5c recDataLength=17 macRecCount=4
decryptRecCount=4.
1: TINYSEC: getTransmitByte() byte=15 data=1
0: TINYSEC: byteReceived() RxByteCnt=13 data=d6 recDataLength=17 macRecCount=5
decryptRecCount=5.
1: TINYSEC: getTransmitByte() byte=16 data=63
0: TINYSEC: byteReceived() RxByteCnt=14 data=db recDataLength=17 macRecCount=6
decryptRecCount=6.
1: TINYSEC: getTransmitByte() byte=17 data=c0
0: TINYSEC: byteReceived() RxByteCnt=15 data=1 recDataLength=17 macRecCount=7

```



```

decryptRecCount=7.
1: TINYSEC: getTransmitByte() byte=18 data=46
0: TINYSEC: byteReceived() RxByteCnt=16 data=63 recDataLength=17 macRecCount=0
decryptRecCount=8.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State GENERAL
1: TINYSEC: getTransmitByte() byte=19 data=7b
0: TINYSEC: byteReceived() RxByteCnt=17 data=c0 recDataLength=17 macRecCount=1
decryptRecCount=9.
1: TINYSEC: getTransmitByte() byte=20 data=d5
0: TINYSEC: byteReceived() RxByteCnt=18 data=46 recDataLength=17 macRecCount=2
decryptRecCount=10.
1: TINYSEC: getTransmitByte() byte=21 data=7d
0: TINYSEC: byteReceived() RxByteCnt=19 data=7b recDataLength=17 macRecCount=3
decryptRecCount=11.
1: TINYSEC: getTransmitByte() byte=22 data=16
0: TINYSEC: byteReceived() RxByteCnt=20 data=d5 recDataLength=17 macRecCount=4
decryptRecCount=12.
1: TINYSEC: getTransmitByte() byte=23 data=ee
0: TINYSEC: byteReceived() RxByteCnt=21 data=7d recDataLength=17 macRecCount=5
decryptRecCount=13.
1: TINYSEC: getTransmitByte() byte=24 data=62
0: TINYSEC: byteReceived() RxByteCnt=22 data=16 recDataLength=17 macRecCount=6
decryptRecCount=14.
1: TINYSEC: getTransmitByte() byte=25 data=28
0: TINYSEC: byteReceived() RxByteCnt=23 data=ee recDataLength=17 macRecCount=7
decryptRecCount=15.
1: TINYSEC: getTransmitByte() byte=26 data=c2
0: TINYSEC: byteReceived() RxByteCnt=24 data=62 recDataLength=17 macRecCount=0
decryptRecCount=16.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State 2LEFTA
0: DBCModeM: ** Switched to state 2LEFTB with 1 remaining
1: TINYSEC: getTransmitByte() byte=27 data=b4
0: TINYSEC: byteReceived() RxByteCnt=25 data=28 recDataLength=17 macRecCount=1
decryptRecCount=17.
0: CBCModeM:incrementalDecrypt: <entry>length 1
0: CBCModeM: incrementalDecrypt. State 2LEFTB
0: incrementalDecrypt: 0 1
1: TINYSEC: getTransmitByte() byte=28 data=45

```

```

0: TINYSEC: byteReceived() RxByteCnt=26 data=c2 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=29 data=62
1: TINYSEC: getTransmitByte() signaling send done
1: MicaHighSpeedRadioTinySecM: Transmit done.
0: TINYSEC: byteReceived() RxByteCnt=27 data=b4 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=28 data=45 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=29 data=62 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: MAC computed: c2 b4 45 62, MAC received: c2 b4 45 62
1: Rcvd ACK 1
1: TestM1M: Sending testdata complete 0x1
1: gent 1 dequeued.
0: TestM1M Timer fired
0: TestM1M: Sending testdata
0: QueuedSend: queue msg enq 1 deq 1
0: QueuedSend: Successfully queued msg to 0x7e, enq 2, deq 1
0: QueuedSend: sending msg (0x1)
0: Enqueueing uart_send_event at 0:0:9.67748925 for mote 00: TestM1M: Sending
testdata complete 0x1
0: gent 1 dequeued.
0: UART: packet transfer complete.
1: TestM1M Timer fired
1: TestM1M: Sending testdata
1: QueuedSend: queue msg enq 2 deq 2
1: QueuedSend: Successfully queued msg to 0xffff, enq 3, deq 2
1: QueuedSend: sending msg (0x2)
1: TINYSEC: sendInit() Sending length = 17
1: TINYSEC: sendInit() Length field = 209
1: bc: 8
1: CBC.encrypt j: 1; bc: 8
1: MAC: computed: 6e 8c cc ef
1: TINYSEC: getTransmitByte() byte=1 data=ff
1: TINYSEC: getTransmitByte() byte=2 data=ff
1: TINYSEC: getTransmitByte() byte=3 data=14
0: Header received byte 1
0: TINYSEC: byteReceived() RxByteCnt=1 data=ff recDataLength=17 macRecCount=1
decryptRecCount=-7.
1: TINYSEC: getTransmitByte() byte=4 data=d1
0: Header received byte 2

```

```

0: TINYSEC: byteReceived() RxByteCnt=2 data=ff recDataLength=17 macRecCount=2
decryptRecCount=-6.
1: TINYSEC: getTransmitByte() byte=5 data=2
0: Header received byte 3
0: TINYSEC: byteReceived() RxByteCnt=3 data=14 recDataLength=17 macRecCount=3
decryptRecCount=-5.
1: TINYSEC: getTransmitByte() byte=6 data=0
0: Header received byte 4
0: TINYSEC: byteReceived() RxByteCnt=4 data=d1 recDataLength=17 macRecCount=4
decryptRecCount=-4.
0: Detected TinySec bit.
0: Encryption enabled.
0: Receive init done TinySec.
0: MAC init called: rxlength=25.
1: TINYSEC: getTransmitByte() byte=7 data=1
0: TINYSEC: byteReceived() RxByteCnt=5 data=2 recDataLength=17 macRecCount=5
decryptRecCount=-3.
1: TINYSEC: getTransmitByte() byte=8 data=0
0: TINYSEC: byteReceived() RxByteCnt=6 data=0 recDataLength=17 macRecCount=6
decryptRecCount=-2.
1: TINYSEC: getTransmitByte() byte=9 data=5a
0: TINYSEC: byteReceived() RxByteCnt=7 data=1 recDataLength=17 macRecCount=7
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=10 data=46
0: TINYSEC: byteReceived() RxByteCnt=8 data=0 recDataLength=17 macRecCount=0
decryptRecCount=0.
0: DECRYPT: init size=17
1: TINYSEC: getTransmitByte() byte=11 data=2b
0: TINYSEC: byteReceived() RxByteCnt=9 data=5a recDataLength=17 macRecCount=1
decryptRecCount=1.
1: TINYSEC: getTransmitByte() byte=12 data=e5
0: TINYSEC: byteReceived() RxByteCnt=10 data=46 recDataLength=17 macRecCount=2
decryptRecCount=2.
1: TINYSEC: getTransmitByte() byte=13 data=5b
0: TINYSEC: byteReceived() RxByteCnt=11 data=2b recDataLength=17 macRecCount=3
decryptRecCount=3.
1: TINYSEC: getTransmitByte() byte=14 data=de
0: TINYSEC: byteReceived() RxByteCnt=12 data=e5 recDataLength=17 macRecCount=4
decryptRecCount=4.
1: TINYSEC: getTransmitByte() byte=15 data=f
0: TINYSEC: byteReceived() RxByteCnt=13 data=5b recDataLength=17 macRecCount=5
decryptRecCount=5.

```

```

1: TINYSEC: getTransmitByte() byte=16 data=1
0: TINYSEC: byteReceived() RxByteCnt=14 data=de recDataLength=17 macRecCount=6
decryptRecCount=6.
1: TINYSEC: getTransmitByte() byte=17 data=79
0: TINYSEC: byteReceived() RxByteCnt=15 data=f recDataLength=17 macRecCount=7
decryptRecCount=7.
1: TINYSEC: getTransmitByte() byte=18 data=3
0: TINYSEC: byteReceived() RxByteCnt=16 data=1 recDataLength=17 macRecCount=0
decryptRecCount=8.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State GENERAL
1: TINYSEC: getTransmitByte() byte=19 data=4c
0: TINYSEC: byteReceived() RxByteCnt=17 data=79 recDataLength=17 macRecCount=1
decryptRecCount=9.
1: TINYSEC: getTransmitByte() byte=20 data=6c
0: TINYSEC: byteReceived() RxByteCnt=18 data=3 recDataLength=17 macRecCount=2
decryptRecCount=10.
1: TINYSEC: getTransmitByte() byte=21 data=f0
0: TINYSEC: byteReceived() RxByteCnt=19 data=4c recDataLength=17 macRecCount=3
decryptRecCount=11.
1: TINYSEC: getTransmitByte() byte=22 data=13
0: TINYSEC: byteReceived() RxByteCnt=20 data=6c recDataLength=17 macRecCount=4
decryptRecCount=12.
1: TINYSEC: getTransmitByte() byte=23 data=e0
0: TINYSEC: byteReceived() RxByteCnt=21 data=f0 recDataLength=17 macRecCount=5
decryptRecCount=13.
1: TINYSEC: getTransmitByte() byte=24 data=0
0: TINYSEC: byteReceived() RxByteCnt=22 data=13 recDataLength=17 macRecCount=6
decryptRecCount=14.
1: TINYSEC: getTransmitByte() byte=25 data=20
0: TINYSEC: byteReceived() RxByteCnt=23 data=e0 recDataLength=17 macRecCount=7
decryptRecCount=15.
1: TINYSEC: getTransmitByte() byte=26 data=6e
0: TINYSEC: byteReceived() RxByteCnt=24 data=0 recDataLength=17 macRecCount=0
decryptRecCount=16.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State 2LEFTA
0: DBCModeM: ** Switched to state 2LEFTB with 1 remaining
1: TINYSEC: getTransmitByte() byte=27 data=8c
0: TINYSEC: byteReceived() RxByteCnt=25 data=20 recDataLength=17 macRecCount=1

```

```

decryptRecCount=17.
0: CBCModeM:incrementalDecrypt: <entry>length 1
0: CBCModeM: incrementalDecrypt. State 2LEFTB
0: incrementalDecrypt: 0 1
1: TINYSEC: getTransmitByte() byte=28 data=cc
0: TINYSEC: byteReceived() RxByteCnt=26 data=6e recDataLength=17 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=29 data=ef
1: TINYSEC: getTransmitByte() signaling send done
1: MicaHighSpeedRadioTinySecM: Transmit done.
0: TINYSEC: byteReceived() RxByteCnt=27 data=8c recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=28 data=cc recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=29 data=ef recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: MAC computed: 6e 8c cc ef, MAC received: 6e 8c cc ef
1: Rcvd ACK 1
1: TestM1M: Sending testdata complete 0x1
1: qent 2 dequeued.
0: TestM1M Timer fired
0: TestM1M: Sending testdata
0: QueuedSend: queue msg enq 2 deq 2
0: QueuedSend: Successfully queued msg to 0x7e, enq 3, deq 2
0: QueuedSend: sending msg (0x2)
0: Enqueueing uart_send_event at 0:0:14.55998925 for mote 00: TestM1M: Sending
testdata complete 0x1
0: qent 2 dequeued.
0: UART: packet transfer complete.
1: TestM1M Timer fired
1: TestM1M: Sending testdata
1: QueuedSend: queue msg enq 3 deq 3
1: QueuedSend: Successfully queued msg to 0xffff, enq 4, deq 3
1: QueuedSend: sending msg (0x3)
1: TINYSEC: sendInit() Sending length = 17
1: TINYSEC: sendInit() Length field = 209
1: bc: 8
1: CBC.encrypt j: 1; bc: 8
1: MAC: computed: f0 c7 2 3a
1: TINYSEC: getTransmitByte() byte=1 data=ff
1: TINYSEC: getTransmitByte() byte=2 data=ff
1: TINYSEC: getTransmitByte() byte=3 data=14

```

```

0: Header received byte 1
0: TINYSEC: byteReceived() RxByteCnt=1 data=ff recDataLength=17 macRecCount=1
decryptRecCount=-7.
1: TINYSEC: getTransmitByte() byte=4 data=d1
0: Header received byte 2
0: TINYSEC: byteReceived() RxByteCnt=2 data=ff recDataLength=17 macRecCount=2
decryptRecCount=-6.
1: TINYSEC: getTransmitByte() byte=5 data=3
0: Header received byte 3
0: TINYSEC: byteReceived() RxByteCnt=3 data=14 recDataLength=17 macRecCount=3
decryptRecCount=-5.
1: TINYSEC: getTransmitByte() byte=6 data=0
0: Header received byte 4
0: TINYSEC: byteReceived() RxByteCnt=4 data=d1 recDataLength=17 macRecCount=4
decryptRecCount=-4.
0: Detected TinySec bit.
0: Encryption enabled.
0: Receive init done TinySec.
0: MAC init called: rxlength=25.
1: TINYSEC: getTransmitByte() byte=7 data=1
0: TINYSEC: byteReceived() RxByteCnt=5 data=3 recDataLength=17 macRecCount=5
decryptRecCount=-3.
1: TINYSEC: getTransmitByte() byte=8 data=0
0: TINYSEC: byteReceived() RxByteCnt=6 data=0 recDataLength=17 macRecCount=6
decryptRecCount=-2.
1: TINYSEC: getTransmitByte() byte=9 data=14
0: TINYSEC: byteReceived() RxByteCnt=7 data=1 recDataLength=17 macRecCount=7
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=10 data=e9
0: TINYSEC: byteReceived() RxByteCnt=8 data=0 recDataLength=17 macRecCount=0
decryptRecCount=0.
0: DECRYPT: init size=17
1: TINYSEC: getTransmitByte() byte=11 data=ca
0: TINYSEC: byteReceived() RxByteCnt=9 data=14 recDataLength=17 macRecCount=1
decryptRecCount=1.
1: TINYSEC: getTransmitByte() byte=12 data=fa
0: TINYSEC: byteReceived() RxByteCnt=10 data=e9 recDataLength=17 macRecCount=2
decryptRecCount=2.
1: TINYSEC: getTransmitByte() byte=13 data=f6
0: TINYSEC: byteReceived() RxByteCnt=11 data=ca recDataLength=17 macRecCount=3
decryptRecCount=3.
1: TINYSEC: getTransmitByte() byte=14 data=fa

```

```

0: TINYSEC: byteReceived() RxByteCnt=12 data=fa recDataLength=17 macRecCount=4
decryptRecCount=4.
1: TINYSEC: getTransmitByte() byte=15 data=97
0: TINYSEC: byteReceived() RxByteCnt=13 data=f6 recDataLength=17 macRecCount=5
decryptRecCount=5.
1: TINYSEC: getTransmitByte() byte=16 data=ca
0: TINYSEC: byteReceived() RxByteCnt=14 data=fa recDataLength=17 macRecCount=6
decryptRecCount=6.
1: TINYSEC: getTransmitByte() byte=17 data=37
0: TINYSEC: byteReceived() RxByteCnt=15 data=97 recDataLength=17 macRecCount=7
decryptRecCount=7.
1: TINYSEC: getTransmitByte() byte=18 data=ac
0: TINYSEC: byteReceived() RxByteCnt=16 data=ca recDataLength=17 macRecCount=0
decryptRecCount=8.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State GENERAL
1: TINYSEC: getTransmitByte() byte=19 data=ad
0: TINYSEC: byteReceived() RxByteCnt=17 data=37 recDataLength=17 macRecCount=1
decryptRecCount=9.
1: TINYSEC: getTransmitByte() byte=20 data=73
0: TINYSEC: byteReceived() RxByteCnt=18 data=ac recDataLength=17 macRecCount=2
decryptRecCount=10.
1: TINYSEC: getTransmitByte() byte=21 data=5d
0: TINYSEC: byteReceived() RxByteCnt=19 data=ad recDataLength=17 macRecCount=3
decryptRecCount=11.
1: TINYSEC: getTransmitByte() byte=22 data=37
0: TINYSEC: byteReceived() RxByteCnt=20 data=73 recDataLength=17 macRecCount=4
decryptRecCount=12.
1: TINYSEC: getTransmitByte() byte=23 data=78
0: TINYSEC: byteReceived() RxByteCnt=21 data=5d recDataLength=17 macRecCount=5
decryptRecCount=13.
1: TINYSEC: getTransmitByte() byte=24 data=cb
0: TINYSEC: byteReceived() RxByteCnt=22 data=37 recDataLength=17 macRecCount=6
decryptRecCount=14.
1: TINYSEC: getTransmitByte() byte=25 data=8
0: TINYSEC: byteReceived() RxByteCnt=23 data=78 recDataLength=17 macRecCount=7
decryptRecCount=15.
1: TINYSEC: getTransmitByte() byte=26 data=f0
0: TINYSEC: byteReceived() RxByteCnt=24 data=cb recDataLength=17 macRecCount=0
decryptRecCount=16.
0: CBCModeM:incrementalDecrypt: <entry>length 8

```

```

0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State 2LEFTA
0: DBCModeM: ** Switched to state 2LEFTB with 1 remaining
1: TINYSEC: getTransmitByte() byte=27 data=c7
0: TINYSEC: byteReceived() RxByteCnt=25 data=8 recDataLength=17 macRecCount=1
decryptRecCount=17.
0: CBCModeM:incrementalDecrypt: <entry>length 1
0: CBCModeM: incrementalDecrypt. State 2LEFTB
0: incrementalDecrypt: 0 1
1: TINYSEC: getTransmitByte() byte=28 data=2
0: TINYSEC: byteReceived() RxByteCnt=26 data=f0 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=29 data=3a
1: TINYSEC: getTransmitByte() signaling send done
1: MicaHighSpeedRadioTinySecM: Transmit done.
0: TINYSEC: byteReceived() RxByteCnt=27 data=c7 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=28 data=2 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=29 data=3a recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: MAC computed: f0 c7 2 3a, MAC received: f0 c7 2 3a
1: Rcvd ACK 1
1: TestM1M: Sending testdata complete 0x1
1: gent 3 dequeued.
0: TestM1M Timer fired
0: TestM1M: Sending testdata
0: QueuedSend: queue msg enq 3 deq 3
0: QueuedSend: Successfully queued msg to 0x7e, enq 4, deq 3
0: QueuedSend: sending msg (0x3)
0: Enqueueing uart_send_event at 0:0:19.44248925 for mote 00: TestM1M: Sending
testdata complete 0x1
0: gent 3 dequeued.
0: UART: packet transfer complete.
1: TestM1M Timer fired
1: TestM1M: Sending testdata
1: QueuedSend: queue msg enq 4 deq 4
1: QueuedSend: Successfully queued msg to 0xffff, enq 5, deq 4
1: MultiHopLEPSM timer task.
1: addr prnt misd rcvd lstS hop rEst sEst
1: 0 126 0 1 0 0 0 0
1: QueuedSend: sending msg (0x4)

```



```

1: MultiHopLEPSM Sending route update msg.
1: QueuedSend: queue msg enq 5 deq 4
1: QueuedSend: Successfully queued msg to 0xffff, enq 6, deq 4
1: TINYSEC: sendInit() Sending length = 17
1: TINYSEC: sendInit() Length field = 209
1: bc: 8
1: CBC.encrypt j: 1; bc: 8
1: MAC: computed: d6 cc 32 ad
1: TINYSEC: getTransmitByte() byte=1 data=ff
1: TINYSEC: getTransmitByte() byte=2 data=ff
1: TINYSEC: getTransmitByte() byte=3 data=14
0: Header received byte 1
0: TINYSEC: byteReceived() RxByteCnt=1 data=ff recDataLength=17 macRecCount=1
decryptRecCount=-7.
1: TINYSEC: getTransmitByte() byte=4 data=d1
0: Header received byte 2
0: TINYSEC: byteReceived() RxByteCnt=2 data=ff recDataLength=17 macRecCount=2
decryptRecCount=-6.
1: TINYSEC: getTransmitByte() byte=5 data=4
0: Header received byte 3
0: TINYSEC: byteReceived() RxByteCnt=3 data=14 recDataLength=17 macRecCount=3
decryptRecCount=-5.
1: TINYSEC: getTransmitByte() byte=6 data=0
0: Header received byte 4
0: TINYSEC: byteReceived() RxByteCnt=4 data=d1 recDataLength=17 macRecCount=4
decryptRecCount=-4.
0: Detected TinySec bit.
0: Encryption enabled.
0: Receive init done TinySec.
0: MAC init called: rxlength=25.
1: TINYSEC: getTransmitByte() byte=7 data=1
0: TINYSEC: byteReceived() RxByteCnt=5 data=4 recDataLength=17 macRecCount=5
decryptRecCount=-3.
1: TINYSEC: getTransmitByte() byte=8 data=0
0: TINYSEC: byteReceived() RxByteCnt=6 data=0 recDataLength=17 macRecCount=6
decryptRecCount=-2.
1: TINYSEC: getTransmitByte() byte=9 data=f4
0: TINYSEC: byteReceived() RxByteCnt=7 data=1 recDataLength=17 macRecCount=7
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=10 data=af
0: TINYSEC: byteReceived() RxByteCnt=8 data=0 recDataLength=17 macRecCount=0
decryptRecCount=0.

```

```

0: DECRYPT: init size=17
1: TINYSEC: getTransmitByte() byte=11 data=e
0: TINYSEC: byteReceived() RxByteCnt=9 data=f4 recDataLength=17 macRecCount=1
decryptRecCount=1.
1: TINYSEC: getTransmitByte() byte=12 data=b4
0: TINYSEC: byteReceived() RxByteCnt=10 data=af recDataLength=17 macRecCount=2
decryptRecCount=2.
1: TINYSEC: getTransmitByte() byte=13 data=30
0: TINYSEC: byteReceived() RxByteCnt=11 data=e recDataLength=17 macRecCount=3
decryptRecCount=3.
1: TINYSEC: getTransmitByte() byte=14 data=f2
0: TINYSEC: byteReceived() RxByteCnt=12 data=b4 recDataLength=17 macRecCount=4
decryptRecCount=4.
1: TINYSEC: getTransmitByte() byte=15 data=fd
0: TINYSEC: byteReceived() RxByteCnt=13 data=30 recDataLength=17 macRecCount=5
decryptRecCount=5.
1: TINYSEC: getTransmitByte() byte=16 data=79
0: TINYSEC: byteReceived() RxByteCnt=14 data=f2 recDataLength=17 macRecCount=6
decryptRecCount=6.
1: TINYSEC: getTransmitByte() byte=17 data=d7
0: TINYSEC: byteReceived() RxByteCnt=15 data=fd recDataLength=17 macRecCount=7
decryptRecCount=7.
1: TINYSEC: getTransmitByte() byte=18 data=ea
0: TINYSEC: byteReceived() RxByteCnt=16 data=79 recDataLength=17 macRecCount=8
decryptRecCount=8.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State GENERAL
1: TINYSEC: getTransmitByte() byte=19 data=69
0: TINYSEC: byteReceived() RxByteCnt=17 data=d7 recDataLength=17 macRecCount=1
decryptRecCount=9.
1: TINYSEC: getTransmitByte() byte=20 data=3d
0: TINYSEC: byteReceived() RxByteCnt=18 data=ea recDataLength=17 macRecCount=2
decryptRecCount=10.
1: TINYSEC: getTransmitByte() byte=21 data=9b
0: TINYSEC: byteReceived() RxByteCnt=19 data=69 recDataLength=17 macRecCount=3
decryptRecCount=11.
1: TINYSEC: getTransmitByte() byte=22 data=3f
0: TINYSEC: byteReceived() RxByteCnt=20 data=3d recDataLength=17 macRecCount=4
decryptRecCount=12.
1: TINYSEC: getTransmitByte() byte=23 data=12
0: TINYSEC: byteReceived() RxByteCnt=21 data=9b recDataLength=17 macRecCount=5

```

```

decryptRecCount=13.
1: TINYSEC: getTransmitByte() byte=24 data=78
0: TINYSEC: byteReceived() RxByteCnt=22 data=3f recDataLength=17 macRecCount=6
decryptRecCount=14.
1: TINYSEC: getTransmitByte() byte=25 data=cc
0: TINYSEC: byteReceived() RxByteCnt=23 data=12 recDataLength=17 macRecCount=7
decryptRecCount=15.
1: TINYSEC: getTransmitByte() byte=26 data=d6
0: TINYSEC: byteReceived() RxByteCnt=24 data=78 recDataLength=17 macRecCount=0
decryptRecCount=16.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State 2LEFTA
0: DBCModeM: ** Switched to state 2LEFTB with 1 remaining
1: TINYSEC: getTransmitByte() byte=27 data=cc
0: TINYSEC: byteReceived() RxByteCnt=25 data=cc recDataLength=17 macRecCount=1
decryptRecCount=17.
0: CBCModeM:incrementalDecrypt: <entry>length 1
0: CBCModeM: incrementalDecrypt. State 2LEFTB
0: incrementalDecrypt: 0 1
1: TINYSEC: getTransmitByte() byte=28 data=32
0: TINYSEC: byteReceived() RxByteCnt=26 data=d6 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=29 data=ad
1: TINYSEC: getTransmitByte() signaling send done
1: MicaHighSpeedRadioTinySecM: Transmit done.
0: TINYSEC: byteReceived() RxByteCnt=27 data=cc recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=28 data=32 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=29 data=ad recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: MAC computed: d6 cc 32 ad, MAC received: d6 cc 32 ad
1: Rcvd ACK 1
1: TestM1M: Sending testdata complete 0x1
.
.
.
.
.
.
1: TestM1M Timer fired

```

```

1: TestMIM: Sending testdata
1: QueuedSend: queue msg enq 10 deq 10
1: QueuedSend: Successfully queued msg to 0x0, enq 11, deq 10
1: MultiHopLEPSM timer task.
1: addr prnt misd rcvd lstS hop rEst sEst
1: 0 126 0 1 14 0 251 255
1: MultiHopLEPSM: Parent = 0
1: QueuedSend: sending msg (0xa)
1: MultiHopLEPSM Sending route update msg.
1: QueuedSend: queue msg enq 11 deq 10
1: QueuedSend: Successfully queued msg to 0xffff, enq 12, deq 10
1: TINYSEC: sendInit() Sending length = 17
1: TINYSEC: sendInit() Length field = 209
1: bc: 8
1: CBC.encrypt j: 1; bc: 8
1: MAC: computed: 7f 15 ee 61
1: TINYSEC: getTransmitByte() byte=1 data=0
1: TINYSEC: getTransmitByte() byte=2 data=0
1: TINYSEC: getTransmitByte() byte=3 data=14
0: Header received byte 1
0: TINYSEC: byteReceived() RxByteCnt=1 data=0 recDataLength=17 macRecCount=1
decryptRecCount=-7.
1: TINYSEC: getTransmitByte() byte=4 data=d1
0: Header received byte 2
0: TINYSEC: byteReceived() RxByteCnt=2 data=0 recDataLength=17 macRecCount=2
decryptRecCount=-6.
1: TINYSEC: getTransmitByte() byte=5 data=4a
0: Header received byte 3
0: TINYSEC: byteReceived() RxByteCnt=3 data=14 recDataLength=17 macRecCount=3
decryptRecCount=-5.
1: TINYSEC: getTransmitByte() byte=6 data=0
0: Header received byte 4
0: TINYSEC: byteReceived() RxByteCnt=4 data=d1 recDataLength=17 macRecCount=4
decryptRecCount=-4.
0: Detected TinySec bit.
0: Encryption enabled.
0: Receive init done TinySec.
0: MAC init called: rxlength=25.
1: TINYSEC: getTransmitByte() byte=7 data=1
0: TINYSEC: byteReceived() RxByteCnt=5 data=4a recDataLength=17 macRecCount=5
decryptRecCount=-3.
1: TINYSEC: getTransmitByte() byte=8 data=0

```

```

0: TINYSEC: byteReceived() RxByteCnt=6 data=0 recDataLength=17 macRecCount=6
decryptRecCount=-2.
1: TINYSEC: getTransmitByte() byte=9 data=92
0: TINYSEC: byteReceived() RxByteCnt=7 data=1 recDataLength=17 macRecCount=7
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=10 data=4f
0: TINYSEC: byteReceived() RxByteCnt=8 data=0 recDataLength=17 macRecCount=0
decryptRecCount=0.
0: DECRYPT: init size=17
1: TINYSEC: getTransmitByte() byte=11 data=34
0: TINYSEC: byteReceived() RxByteCnt=9 data=92 recDataLength=17 macRecCount=1
decryptRecCount=1.
1: TINYSEC: getTransmitByte() byte=12 data=c2
0: TINYSEC: byteReceived() RxByteCnt=10 data=4f recDataLength=17 macRecCount=2
decryptRecCount=2.
1: TINYSEC: getTransmitByte() byte=13 data=87
0: TINYSEC: byteReceived() RxByteCnt=11 data=34 recDataLength=17 macRecCount=3
decryptRecCount=3.
1: TINYSEC: getTransmitByte() byte=14 data=d9
0: TINYSEC: byteReceived() RxByteCnt=12 data=c2 recDataLength=17 macRecCount=4
decryptRecCount=4.
1: TINYSEC: getTransmitByte() byte=15 data=f8
0: TINYSEC: byteReceived() RxByteCnt=13 data=87 recDataLength=17 macRecCount=5
decryptRecCount=5.
1: TINYSEC: getTransmitByte() byte=16 data=b5
0: TINYSEC: byteReceived() RxByteCnt=14 data=d9 recDataLength=17 macRecCount=6
decryptRecCount=6.
1: TINYSEC: getTransmitByte() byte=17 data=b1
0: TINYSEC: byteReceived() RxByteCnt=15 data=f8 recDataLength=17 macRecCount=7
decryptRecCount=7.
1: TINYSEC: getTransmitByte() byte=18 data=a
0: TINYSEC: byteReceived() RxByteCnt=16 data=b5 recDataLength=17 macRecCount=0
decryptRecCount=8.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State GENERAL
1: TINYSEC: getTransmitByte() byte=19 data=53
0: TINYSEC: byteReceived() RxByteCnt=17 data=b1 recDataLength=17 macRecCount=1
decryptRecCount=9.
1: TINYSEC: getTransmitByte() byte=20 data=4b
0: TINYSEC: byteReceived() RxByteCnt=18 data=a recDataLength=17 macRecCount=2
decryptRecCount=10.

```

```

1: TINYSEC: getTransmitByte() byte=21 data=2c
0: TINYSEC: byteReceived() RxByteCnt=19 data=53 recDataLength=17 macRecCount=3
decryptRecCount=11.
1: TINYSEC: getTransmitByte() byte=22 data=14
0: TINYSEC: byteReceived() RxByteCnt=20 data=4b recDataLength=17 macRecCount=4
decryptRecCount=12.
1: TINYSEC: getTransmitByte() byte=23 data=17
0: TINYSEC: byteReceived() RxByteCnt=21 data=2c recDataLength=17 macRecCount=5
decryptRecCount=13.
1: TINYSEC: getTransmitByte() byte=24 data=b4
0: TINYSEC: byteReceived() RxByteCnt=22 data=14 recDataLength=17 macRecCount=6
decryptRecCount=14.
1: TINYSEC: getTransmitByte() byte=25 data=5b
0: TINYSEC: byteReceived() RxByteCnt=23 data=17 recDataLength=17 macRecCount=7
decryptRecCount=15.
1: TINYSEC: getTransmitByte() byte=26 data=7f
0: TINYSEC: byteReceived() RxByteCnt=24 data=b4 recDataLength=17 macRecCount=0
decryptRecCount=16.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State 2LEFTA
0: DBCModeM: ** Switched to state 2LEFTB with 1 remaining
1: TINYSEC: getTransmitByte() byte=27 data=15
0: TINYSEC: byteReceived() RxByteCnt=25 data=5b recDataLength=17 macRecCount=1
decryptRecCount=17.
0: CBCModeM:incrementalDecrypt: <entry>length 1
0: CBCModeM: incrementalDecrypt. State 2LEFTB
0: incrementalDecrypt: 0 1
1: TINYSEC: getTransmitByte() byte=28 data=ee
0: TINYSEC: byteReceived() RxByteCnt=26 data=7f recDataLength=17 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=29 data=61
1: TINYSEC: getTransmitByte() signaling send done
1: MicaHighSpeedRadioTinySecM: Transmit done.
0: TINYSEC: byteReceived() RxByteCnt=27 data=15 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=28 data=ee recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=29 data=61 recDataLength=17 macRecCount=-1
decryptRecCount=-1.
0: MAC computed: 7f 15 ee 61, MAC received: 7f 15 ee 61
0: QueuedSend: queue msg enq 17 deq 17

```

```

0: QueuedSend: Successfully queued msg to 0x7e, enq 18, deq 17
0: QueuedSend: sending msg (0x11)
0: Enqueueing uart_send_event at 0:4:56.44994275 for mote 01: Rcvd ACK 1
1: TestM1M: Sending testdata complete 0x1
1: qent 10 dequeued.
1: QueuedSend: sending msg (0xb)
0: qent 17 dequeued.
0: UART: packet transfer complete.
1: TINYSEC: sendInit() Sending length = 13
1: TINYSEC: sendInit() Length field = 205
1: bc: 0
1: CBC.encrypt j: 5; bc: 0
1: MAC: computed: e8 34 f9 d7
1: TINYSEC: getTransmitByte() byte=1 data=ff
1: TINYSEC: getTransmitByte() byte=2 data=ff
1: TINYSEC: getTransmitByte() byte=3 data=fa
0: Header received byte 1
0: TINYSEC: byteReceived() RxByteCnt=1 data=ff recDataLength=17 macRecCount=1
decryptRecCount=-7.
1: TINYSEC: getTransmitByte() byte=4 data=cd
0: Header received byte 2
0: TINYSEC: byteReceived() RxByteCnt=2 data=ff recDataLength=17 macRecCount=2
decryptRecCount=-6.
1: TINYSEC: getTransmitByte() byte=5 data=4b
0: Header received byte 3
0: TINYSEC: byteReceived() RxByteCnt=3 data=fa recDataLength=17 macRecCount=3
decryptRecCount=-5.
1: TINYSEC: getTransmitByte() byte=6 data=0
0: Header received byte 4
0: TINYSEC: byteReceived() RxByteCnt=4 data=cd recDataLength=17 macRecCount=4
decryptRecCount=-4.
0: Detected TinySec bit.
0: Encryption enabled.
0: Receive init done TinySec.
0: MAC init called: rxlength=21.
1: TINYSEC: getTransmitByte() byte=7 data=1
0: TINYSEC: byteReceived() RxByteCnt=5 data=4b recDataLength=13 macRecCount=5
decryptRecCount=-3.
1: TINYSEC: getTransmitByte() byte=8 data=0
0: TINYSEC: byteReceived() RxByteCnt=6 data=0 recDataLength=13 macRecCount=6
decryptRecCount=-2.
1: TINYSEC: getTransmitByte() byte=9 data=78

```

```

0: TINYSEC: byteReceived() RxByteCnt=7 data=1 recDataLength=13 macRecCount=7
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=10 data=be
0: TINYSEC: byteReceived() RxByteCnt=8 data=0 recDataLength=13 macRecCount=0
decryptRecCount=0.
0: DECRYPT: init size=13
1: TINYSEC: getTransmitByte() byte=11 data=c9
0: TINYSEC: byteReceived() RxByteCnt=9 data=78 recDataLength=13 macRecCount=1
decryptRecCount=1.
1: TINYSEC: getTransmitByte() byte=12 data=74
0: TINYSEC: byteReceived() RxByteCnt=10 data=be recDataLength=13 macRecCount=2
decryptRecCount=2.
1: TINYSEC: getTransmitByte() byte=13 data=d7
0: TINYSEC: byteReceived() RxByteCnt=11 data=c9 recDataLength=13 macRecCount=3
decryptRecCount=3.
1: TINYSEC: getTransmitByte() byte=14 data=be
0: TINYSEC: byteReceived() RxByteCnt=12 data=74 recDataLength=13 macRecCount=4
decryptRecCount=4.
1: TINYSEC: getTransmitByte() byte=15 data=96
0: TINYSEC: byteReceived() RxByteCnt=13 data=d7 recDataLength=13 macRecCount=5
decryptRecCount=5.
1: TINYSEC: getTransmitByte() byte=16 data=34
0: TINYSEC: byteReceived() RxByteCnt=14 data=be recDataLength=13 macRecCount=6
decryptRecCount=6.
1: TINYSEC: getTransmitByte() byte=17 data=6e
0: TINYSEC: byteReceived() RxByteCnt=15 data=96 recDataLength=13 macRecCount=7
decryptRecCount=7.
1: TINYSEC: getTransmitByte() byte=18 data=b5
0: TINYSEC: byteReceived() RxByteCnt=16 data=34 recDataLength=13 macRecCount=0
decryptRecCount=8.
0: CBCModeM:incrementalDecrypt: <entry>length 8
0: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
0: CBCModeM: incrementalDecrypt. State 2LEFTA
0: DBCModeM: ** Switched to state 2LEFTB with 5 remaining
1: TINYSEC: getTransmitByte() byte=19 data=1
0: TINYSEC: byteReceived() RxByteCnt=17 data=6e recDataLength=13 macRecCount=1
decryptRecCount=9.
1: TINYSEC: getTransmitByte() byte=20 data=8f
0: TINYSEC: byteReceived() RxByteCnt=18 data=b5 recDataLength=13 macRecCount=2
decryptRecCount=10.
1: TINYSEC: getTransmitByte() byte=21 data=a3
0: TINYSEC: byteReceived() RxByteCnt=19 data=1 recDataLength=13 macRecCount=3

```



```

decryptRecCount=11.
1: TINYSEC: getTransmitByte() byte=22 data=e8
0: TINYSEC: byteReceived() RxByteCnt=20 data=8f recDataLength=13 macRecCount=4
decryptRecCount=12.
1: TINYSEC: getTransmitByte() byte=23 data=34
0: TINYSEC: byteReceived() RxByteCnt=21 data=a3 recDataLength=13 macRecCount=5
decryptRecCount=13.
0: CBCModeM:incrementalDecrypt: <entry>length 5
0: CBCModeM: incrementalDecrypt. State 2LEFTB
0: incrementalDecrypt: 0 5
0: incrementalDecrypt: 1 5
0: incrementalDecrypt: 2 5
0: incrementalDecrypt: 3 5
0: incrementalDecrypt: 4 5
1: TINYSEC: getTransmitByte() byte=24 data=f9
0: TINYSEC: byteReceived() RxByteCnt=22 data=e8 recDataLength=13 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: getTransmitByte() byte=25 data=d7
1: TINYSEC: getTransmitByte() signaling send done
1: MicaHighSpeedRadioTinySecM: Transmit done.
0: TINYSEC: byteReceived() RxByteCnt=23 data=34 recDataLength=13 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=24 data=f9 recDataLength=13 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: byteReceived() RxByteCnt=25 data=d7 recDataLength=13 macRecCount=-1
decryptRecCount=-1.
0: MAC computed: e8 34 f9 d7, MAC received: e8 34 f9 d7
1: Rcvd ACK 1
1: qent 11 dequeued.
0: TestM1M Timer fired
0: TestM1M: Sending testdata
0: QueuedSend: queue msg enq 18 deq 18
0: QueuedSend: Successfully queued msg to 0x7e, enq 19, deq 18
0: MultiHopLEPSM timer task.
0: addr prnt misd rcvd lstS hop rEst sEst
0: 1 0 0 5 75 1 255 251
0: MultiHopLEPSM: Parent = 126
0: QueuedSend: sending msg (0x12)
0: MultiHopLEPSM Sending route update msg.
0: QueuedSend: queue msg enq 19 deq 18
0: QueuedSend: Successfully queued msg to 0xffff, enq 20, deq 18
0: Enqueueing uart_send_event at 0:4:57.74498925 for mote 00: TestM1M: Sending

```

```

testdata complete 0x1
0: qent 18 dequeued.
0: UART: packet transfer complete.
0: QueuedSend: sending msg (0x13)
0: TINYSEC: sendInit() Sending length = 13
0: TINYSEC: sendInit() Length field = 205
0: bc: 0
0: CBC.encrypt j: 5; bc: 0
0: MAC: computed: d0 95 e1 d4
0: TINYSEC: getTransmitByte() byte=1 data=ff
0: TINYSEC: getTransmitByte() byte=2 data=ff
0: TINYSEC: getTransmitByte() byte=3 data=fa
1: Header received byte 1
1: TINYSEC: byteReceived() RxByteCnt=1 data=ff recDataLength=13 macRecCount=1
decryptRecCount=-7.
0: TINYSEC: getTransmitByte() byte=4 data=cd
1: Header received byte 2
1: TINYSEC: byteReceived() RxByteCnt=2 data=ff recDataLength=13 macRecCount=2
decryptRecCount=-6.
0: TINYSEC: getTransmitByte() byte=5 data=f
1: Header received byte 3
1: TINYSEC: byteReceived() RxByteCnt=3 data=fa recDataLength=13 macRecCount=3
decryptRecCount=-5.
0: TINYSEC: getTransmitByte() byte=6 data=0
1: Header received byte 4
1: TINYSEC: byteReceived() RxByteCnt=4 data=cd recDataLength=13 macRecCount=4
decryptRecCount=-4.
1: Detected TinySec bit.
1: Encryption enabled.
1: Receive init done TinySec.
1: MAC init called: rxlength=21.
0: TINYSEC: getTransmitByte() byte=7 data=0
1: TINYSEC: byteReceived() RxByteCnt=5 data=f recDataLength=13 macRecCount=5
decryptRecCount=-3.
0: TINYSEC: getTransmitByte() byte=8 data=0
1: TINYSEC: byteReceived() RxByteCnt=6 data=0 recDataLength=13 macRecCount=6
decryptRecCount=-2.
0: TINYSEC: getTransmitByte() byte=9 data=6c
1: TINYSEC: byteReceived() RxByteCnt=7 data=0 recDataLength=13 macRecCount=7
decryptRecCount=-1.
0: TINYSEC: getTransmitByte() byte=10 data=e2
1: TINYSEC: byteReceived() RxByteCnt=8 data=0 recDataLength=13 macRecCount=0

```

```

decryptRecCount=0.
1: DECRYPT: init size=13
0: TINYSEC: getTransmitByte() byte=11 data=ff
1: TINYSEC: byteReceived() RxByteCnt=9 data=6c recDataLength=13 macRecCount=1
decryptRecCount=1.
0: TINYSEC: getTransmitByte() byte=12 data=8a
1: TINYSEC: byteReceived() RxByteCnt=10 data=e2 recDataLength=13 macRecCount=2
decryptRecCount=2.
0: TINYSEC: getTransmitByte() byte=13 data=e8
1: TINYSEC: byteReceived() RxByteCnt=11 data=ff recDataLength=13 macRecCount=3
decryptRecCount=3.
0: TINYSEC: getTransmitByte() byte=14 data=dd
1: TINYSEC: byteReceived() RxByteCnt=12 data=8a recDataLength=13 macRecCount=4
decryptRecCount=4.
0: TINYSEC: getTransmitByte() byte=15 data=f9
1: TINYSEC: byteReceived() RxByteCnt=13 data=e8 recDataLength=13 macRecCount=5
decryptRecCount=5.
0: TINYSEC: getTransmitByte() byte=16 data=9c
1: TINYSEC: byteReceived() RxByteCnt=14 data=dd recDataLength=13 macRecCount=6
decryptRecCount=6.
0: TINYSEC: getTransmitByte() byte=17 data=3c
1: TINYSEC: byteReceived() RxByteCnt=15 data=f9 recDataLength=13 macRecCount=7
decryptRecCount=7.
0: TINYSEC: getTransmitByte() byte=18 data=5b
1: TINYSEC: byteReceived() RxByteCnt=16 data=9c recDataLength=13 macRecCount=0
decryptRecCount=8.
1: CBCModeM:incrementalDecrypt: <entry>length 8
1: incrementalDecrypt: Moved 8 bytes; 0 remaining this run
1: CBCModeM: incrementalDecrypt. State 2LEFTA
1: DBCModeM: ** Switched to state 2LEFTB with 5 remaining
0: TINYSEC: getTransmitByte() byte=19 data=ca
1: TINYSEC: byteReceived() RxByteCnt=17 data=3c recDataLength=13 macRecCount=1
decryptRecCount=9.
0: TINYSEC: getTransmitByte() byte=20 data=64
1: TINYSEC: byteReceived() RxByteCnt=18 data=5b recDataLength=13 macRecCount=2
decryptRecCount=10.
0: TINYSEC: getTransmitByte() byte=21 data=99
1: TINYSEC: byteReceived() RxByteCnt=19 data=ca recDataLength=13 macRecCount=3
decryptRecCount=11.
0: TINYSEC: getTransmitByte() byte=22 data=d0
1: TINYSEC: byteReceived() RxByteCnt=20 data=64 recDataLength=13 macRecCount=4
decryptRecCount=12.

```

```

0: TINYSEC: getTransmitByte() byte=23 data=95
1: TINYSEC: byteReceived() RxByteCnt=21 data=99 recDataLength=13 macRecCount=5
decryptRecCount=13.
1: CBCModeM:incrementalDecrypt: <entry>length 5
1: CBCModeM: incrementalDecrypt. State 2LEFTB
1: incrementalDecrypt: 0 5
1: incrementalDecrypt: 1 5
1: incrementalDecrypt: 2 5
1: incrementalDecrypt: 3 5
1: incrementalDecrypt: 4 5
0: TINYSEC: getTransmitByte() byte=24 data=e1
1: TINYSEC: byteReceived() RxByteCnt=22 data=d0 recDataLength=13 macRecCount=-1
decryptRecCount=-1.
0: TINYSEC: getTransmitByte() byte=25 data=d4
0: TINYSEC: getTransmitByte() signaling send done
0: MicaHighSpeedRadioTinySecM: Transmit done.
1: TINYSEC: byteReceived() RxByteCnt=23 data=95 recDataLength=13 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: byteReceived() RxByteCnt=24 data=e1 recDataLength=13 macRecCount=-1
decryptRecCount=-1.
1: TINYSEC: byteReceived() RxByteCnt=25 data=d4 recDataLength=13 macRecCount=-1
decryptRecCount=-1.
1: MAC computed: d0 95 e1 d4, MAC received: d0 95 e1 d4
0: Rcvd ACK 1
0: qent 19 dequeued.
Simulation of 2 motes completed.

```

APPENDIX E.2: Test1 'Listen' Tool Output

```
user@awmxdp /opt/tinyos-1.x/tools/java
$ export MOTECOM=tossim-serial
$ java net.tinyos.tools.Listen
```

Opening tossim-serial source

Connecting to Tossim event port at localhost:10585

Connection opened to TOSSIM event port

Connecting to Tossim command port at localhost:10584

Connection opened to Tossim command port

```
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 1A 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 1B 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 1C 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 1D 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 1F 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 20 00 00 01 23 45 67 89 AB CD EF 01 00
```


7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 3B 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 3D 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 3E 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 3F 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 40 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 42 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 43 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 44 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 45 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 47 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 48 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 49 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00
7E 00 14 7D 11 00 00 01 00 4A 00 00 01 23 45 67 89 AB CD EF 01 00
7E 00 14 7D 11 00 00 00 00 00 00 00 01 23 45 67 89 AB CD EF 00 00

Closing tossim-serial source

Error on tossim-serial@localhost: java.io.EOFException

APPENDIX F: Test2 TOSSIM/Simulation Output

SIM: Random seed is 421524

SIM: Time for mote 0 initialized to 16763545.

0: Cipher: Misty1

0: TestM1BM: Timer fired

0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:

0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:

0: TestM1BM: IV = 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:

0: TestM1BM: CIPHERTEXT = 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0:

0: TestM1BM: RECOVERED P-TEXT (pre-enc) = 0: A 0: A 0: A 0: A 0: A 0: A 0: A 0: A 0:

0: TestM1BM: Proceeding to encrypt data

0: TestM1BM: Data encryption done

0: TestM1BM: CIPHERTEXT (post-enc) = 0: - 0: _ 0: X 0: Š 0: ċ 0: n 0: Á 0: 0:

0: TestM1BM: Proceeding to decrypt data

0: TestM1BM: Data decryption done

0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:

0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:

0: TestM1BM: IV = 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:

0: TestM1BM: CIPHERTEXT = 0: - 0: _ 0: X 0: Š 0: ċ 0: n 0: Á 0: 0:

0: TestM1BM: RECOVERED P-TEXT (post-dec) = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:

0: TestM1BM: Timer fired

0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:

0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:

0: TestM1BM: IV = 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:

0: TestM1BM: CIPHERTEXT = 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0:

0: TestM1BM: RECOVERED P-TEXT (pre-enc) = 0: A 0: A 0: A 0: A 0: A 0: A 0: A 0: A 0:

0: TestM1BM: Proceeding to encrypt data

0: TestM1BM: Data encryption done

0: TestM1BM: CIPHERTEXT (post-enc) = 0: " 0: U 0: Ä 0: G 0: " 0: R 0: å 0: > 0:

0: TestM1BM: Proceeding to decrypt data

0: TestM1BM: Data decryption done

0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:

0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:

0: TestM1BM: IV = 0: 0x1 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:

0: TestM1BM: CIPHERTEXT = 0: " 0: U 0: Ä 0: G 0: " 0: R 0: å 0: > 0:

0: TestM1BM: RECOVERED P-TEXT (post-dec) = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:


```

0: TestM1BM: Timer fired
0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:
0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:
0: TestM1BM: IV = 0: 0x1 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:
0: TestM1BM: CIPHERTEXT = 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0:
0: TestM1BM: RECOVERED P-TEXT (pre-enc) = 0: A 0: A 0: A 0: A 0: A 0: A 0: A 0:
0: TestM1BM: Proceeding to encrypt data
0: TestM1BM: Data encryption done
0: TestM1BM: CIPHERTEXT (post-enc) = 0: ¢ 0: c 0: ø 0: ò 0: 3 0: ¼ 0: Å 0: < 0:
0: TestM1BM: Proceeding to decrypt data
0: TestM1BM: Data decryption done
0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:
0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:
0: TestM1BM: IV = 0: 0x2 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:
0: TestM1BM: CIPHERTEXT = 0: ¢ 0: c 0: ø 0: ò 0: 3 0: ¼ 0: Å 0: < 0:
0: TestM1BM: RECOVERED P-TEXT (post-dec) = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:

0: TestM1BM: Timer fired
0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:
0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:
0: TestM1BM: IV = 0: 0x2 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:
0: TestM1BM: CIPHERTEXT = 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0:
0: TestM1BM: RECOVERED P-TEXT (pre-enc) = 0: A 0: A 0: A 0: A 0: A 0: A 0: A 0:
0: TestM1BM: Proceeding to encrypt data
0: TestM1BM: Data encryption done
0: TestM1BM: CIPHERTEXT (post-enc) = 0: 0 0: Õ 0: ã 0: ` 0: É 0: Ü 0: ó 0: o 0:
0: TestM1BM: Proceeding to decrypt data
0: TestM1BM: Data decryption done
0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:
0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:
0: TestM1BM: IV = 0: 0x3 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:
0: TestM1BM: CIPHERTEXT = 0: 0 0: Õ 0: ã 0: ` 0: É 0: Ü 0: ó 0: o 0:
0: TestM1BM: RECOVERED P-TEXT (post-dec) = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:

0: TestM1BM: Timer fired
0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:
0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:

```

```

0: TestM1BM: IV = 0: 0x3 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:
0: TestM1BM: CIPHERTEXT = 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0: Z 0:
0: TestM1BM: RECOVERED P-TEXT (pre-enc) = 0: A 0: A 0: A 0: A 0: A 0: A 0: A 0:
0: TestM1BM: Proceeding to encrypt data
0: TestM1BM: Data encryption done
0: TestM1BM: CIPHERTEXT (post-enc) = 0: ö 0: ý 0: \ 0: ½ 0: ø 0: 1 0: & 0: F 0:
0: TestM1BM: Proceeding to decrypt data
0: TestM1BM: Data decryption done
0: TestM1BM: APP KEY = 0: 0x1 0: 0x23 0: 0x45 0: 0x67 0: 0x89 0: 0xab 0: 0xcd 0: 0xef
0:
0: TestM1BM: SOURCE PLAINTEXT = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:
0: TestM1BM: IV = 0: 0x4 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0: 0x0 0:
0: TestM1BM: CIPHERTEXT = 0: ö 0: ý 0: \ 0: ½ 0: ø 0: 1 0: & 0: F 0:
0: TestM1BM: RECOVERED P-TEXT (post-dec) = 0: T 0: E 0: S 0: T 0: I 0: N 0: G 0: ! 0:
Simulation of 1 notes completed.

```

REFERENCES

- [1] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler and J. D. Tygar, “SPINS: Security Protocols for Sensor Networks”, In The Seventh Annual International Conference on Mobile Computing and Networking (MobiCom 2001), pp 189-199. ACM Press, 2001.
- [2] Kamran Jamshaid, “A framework for implementing security in wireless sensor networks”, Master’s Thesis, Wayne State University, MI, 2002.
- [3] Chris Karlof, Naveen Sastry and David Wagner, “TinySec: A Link Layer Security Architecture for Wireless Sensor Networks”, In Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004), pp.162–175. ACM Press, 2004.
- [4] Chris Karlof, Naveen Sastry and David Wagner, “TinySec: User Manual”, UC Berkeley, June 2004.
- [5] Jonathan Polley, Dionysys Blazakis, Jonathan McGee, Dan Rusk, John S. Baras and Manish Karir, “ATEMU: A Fine-grained Sensor Network Simulator”, In Proceedings of SECON’04, First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, pp 145-152, 2004.
- [6] B. Titzer, D. Lee, and J. Palsberg, “Aurora: Scalable sensor network simulation with precise timing”, in Proc. 4th Int’l Conf. Information Processing Sensor Networks (IPSN ’05). IEEE, Piscataway, NJ, Apr. 2005, pp. 477–482.
- [7] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin, “Emstar: An environment for developing wireless embedded systems software”, Technical Report Technical Report 0009, CENS, Mar. 2003.

- [8] S. Sundresh, W. Kim, and G. Agha, "SENS: A sensor, environment and network simulator", In Proceedings of 37th Annual Simulation Symposium, 2004, pp. 221–230.
- [9] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications", In Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003). ACM Press, New York, Nov. 2003, pp. 126–137.
- [10] P. Levis and N. Lee, "TOSSIM: A Simulator for TinyOS Networks - Version 1.0 June 26, 2003", September 2003.
- [11] David Noete, "Wireless Sensor Network Applications: Blimps Project Report", April 2007.
- [12] T. Li, H. Wu and F. Bao, "SenSec Design", Institute for InfoComm Research, Tech. Rep. TR-I2R-v1.1, 2005.
- [13] Xinkai Wang, Tieyan Li and Hongjun Wu, "SenSec User Manual", February 2005.
- [14] Alec Woo, Terence Tong and David Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks", The First ACM Conference on Embedded Networked Sensor Systems (SenSys2003).
- [15] D. Gay, P. Levis, J. R. von Behren, M. Welsh, E. A. Brewer, and D. E. Culler, "The nesC language: A holistic approach to networked embedded systems," In Proceedings of PLDI'03, ACM SIGPLAN Conference on Programming Language Design and Implementation, 2003, pp. 1–11.
- [16] David Gay, Philip Levis, David Culler and Eric Brewer, "nesC 1.1 Language Reference Manual", May 2003

- [17] National Institute of Standards and Technology (NIST), “SKIPJACK and KEA Algorithm Specifications”, May 1998 and “Clarification to the Skipjack Algorithm Specification”, May 9, 2002. <http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf>
- [18] H. Ohta and M. Matsui, “Internet informational RFC2994: A Description of the MISTY1 Encryption Algorithm”, Nov 2000, <http://tools.ietf.org/html/rfc2994>
- [19] Hironobu Suzuki, “Optimized C code modules for gcc-2.95 Misty1”, <http://www.pp.iij4u.or.jp/~h2np/archives/misty1-gcc2.95.tar.gz>
- [20] Johann Großschädl, “TinySA: A Security Architecture for Wireless Sensor Networks (Extended Abstract)”, In Proceedings of the 2nd International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2006), ACM Press, 2006.
- [21] Carl Hartung, James Balsalle and Richard Han. “Node Compromise in Sensor Networks: The Need for Secure Systems”, University of Colorado Technical Report. CU-CS-990-05. December 2004.
- [22] National Institute of Standards and Technology (NIST), “Federal Information Processing Standards (FIPS) Publication 197: ADVANCED ENCRYPTION STANDARD (AES aka Rijndael)”, November 2001, <http://csrc.nist.gov/publications/>
- [23] J. Daemen and V. Rijmen, “AES Proposal: Rijndael (Version 2)”, NIST AES website 1999, csrc.nist.gov/encryption/aes
- [24] Laurent Haan, “AES tutorial and C implementation”, <http://www.progressive-coding.com/tutorial.php?id=0>
- [25] C. Karlof and D. Wagner. “Secure routing in wireless sensor networks: Attacks and countermeasures”, In First IEEE International Workshop on Sensor Network Protocols and Applications, pp 113-127, May 2003.

- [26] Jeff Thorn, "Octave Tech Brief #5-01: Deciphering TinyOS Serial Packets", March 2005
- [27] M Healy, T Newe and E Lewis, "Efficiently securing data on a wireless sensor network", In Journal of Physics: Conference Series, Vol 76, page: 012063, 2007.
- [28] Junko Nakajima, Tetsuya Ichikawa and Tomomi Kasuya, "Cipher Algorithm Implementation", In Mitsubishi Electric Advance Technical Report, pp 13-17, December 2002.
- [29] M. Matsui, "New Block Encryption Algorithm MISTY", In Proceedings of Fourth International Workshop of Fast Software Encryption, pp 54-86, Jan 1997.
- [30] M. Matsui, "Supporting Document of MISTY1 - Version 1.10", Mitsubishi Electric Corp, September 2000.
- [31] TinyOS, <http://www.tinyos.net/>
- [32] TinyOS Help, <http://mail.millennium.berkeley.edu/pipermail/tinyos-help/>
- [33] TinyOS Sourceforge, <http://sourceforge.net/projects/tinyos/>
- [34] nesC Sourceforge, <http://nescs.sourceforge.net>
- [35] Hints on using hardware security in CC2420
<http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2006-August/018509.html>
- [36] Crossbow, <http://www.xbow.com/Home/HomePage.aspx>
- [37] http://en.wikipedia.org/wiki/Wireless_sensor_network
- [38] <http://en.wikipedia.org/wiki/Cryptography>

[39] [http://en.wikipedia.org/wiki/Skipjack_\(cipher\)](http://en.wikipedia.org/wiki/Skipjack_(cipher))

[40] <http://en.wikipedia.org/wiki/MISTY1>

[41] http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[42] AES, <http://www.iaik.tu-graz.ac.at/research/krypto/AES/old/%7Erijmen/rijndael/>

[43] Programmer's Notepad, <http://www.pnotepad.org/>

[44] Cygwin, <http://www.cygwin.com/>

Graphic sources:

- Figure 1: Crossbow and SUN SPOT websites
- Figures 3-7: Wikipedia
- Figure 8: [3]

2010-1