# μSec: A Security Protocol for Unicast Communication in Wireless Sensor Networks

Amrita Ghosal, Sanjib Sur, and Sipra DasBit

Department of Computer Science and Technology
Bengal Engineering and Science University, Shibpur, Howrah-711103, India
ghosal_amrita@yahoo.com, as-sanjib.sur11@gmail.com
siprad@hotmail.com

**Abstract.** Secure communication in wireless sensor networks (WSNs) not only needs to provide the basic security but also to defend various attacks. The challenge in providing security in this network is that the securing mechanism must be lightweight to make it implementable in resource-constrained nodes. In this paper we have devised a link layer protocol for securing unicast communication in wireless sensor network (WSN). The protocol (μSec) is developed in TinyOS platform which is an event-driven operating system used in WSN for networked applications. Our protocol supports the basic security features such as confidentiality, authentication and integrity along with defense against replay attacks. We have modified an existing cryptographic algorithm with a target to minimize computational overhead to make it implementable in WSN. A simple, counter based defense mechanism is proposed to thwart replay attack. Both qualitative and quantitative analyses are performed to measure the efficacy of the protocol. The protocol is compared with some of important security protocols developed around TinyOS. We claim that that μSec, in addition to basic security, thwarts replay attack with same overhead as in other protocols which have considered basic securities only. We further claim that the μSec requires 10% (avg.) less overhead compared to its competitor which also defends replay attack.

**Keywords:** Wireless sensor networks, Sensor Network Security, Authentication, Encryption, Unicast communication.

## 1 Introduction

The advent of efficient short range radio communication and advances in miniaturization of computing devices have made possible the development of wireless sensor networks (WSNs). The WSNs have gained popularity due to the fact that they are potentially low-cost solutions to a variety of real-world challenges. This network consists of a large number of small, battery-powered wireless sensor nodes where the nodes are usually equipped with hardware components such as sensing unit, processing unit with small memory, transceiver unit, power unit and a small piece of software e.g. TinyOS [1]. The TinyOS is an event-driven operating system for networked applications in wireless embedded systems. The WSN also has a base station

R. Di Pietro et al. (Eds.): DPM 2012 and SETOP 2012, LNCS 7731, pp. 258–273, 2013.

which is a high-end node compared to sensor nodes and acts as the network's interface with the outside world. The sensor nodes are responsible for gathering data from the environment and send data to the base station. The base station further processes the data before transmitting to the outside world.

WSNs are vulnerable to attacks that are more difficult from being launched in wired networks [2]. So, the need of ensuring security while communicating in presence of such attack is of utmost importance. An attack is defined as an attempt to gain unauthorized access to a service, resource, or information to compromise integrity, availability, or confidentiality. Replay attack is one such attack where packets are captured and replayed into the network by an adversary [3]. Further, one important difference of WSNs from other conventional wireless networks is dealing with extreme constrained resources in terms of battery power, bandwidth, processing capability and storage. Therefore, in order to consider such resource constraints it is desirable for providing reasonable security strength while limiting overhead. Traditional security protocols are difficult for implementation in sensor networks as they require more powerful resources which a WSN cannot provide.

Many works have been reported so far that have contributed towards securing communication of WSN. A few of them, however, has been developed [4-8] on TinyOS platform where the security features are added to TinyOS. Authors in [4] proposed TinySec, the first link layer security protocol which actually adds security feature to the TinyOS environment that lacked the inherent security properties. It covers the basic security needs such as message authenticity, integrity, and confidentiality. TinySec supports two different security options: authenticated encryption (TinySec-AE) and authentication only (TinySec-Auth). With authenticated encryption, TinySec encrypts the data and authenticates the packet with a MAC. The MAC is computed over the encrypted payload and the packet header. In authentication only mode, it authenticates the entire packet with a MAC, but the data is not encrypted. Here encryption is done using CBC mode for encryption. TinySec has been implemented in testbed using Berkeley sensor nodes and also has been simulated in TOSSIM simulator. But the drawback of the security feature of TinySec is that it is not strong enough as it employs a single network-wide key for all the nodes, such that every node in the network can impersonate as any other node. Also TinySec does not address any explicit attack e.g. replay attack.

In [5] the authors have developed a security protocol SNEP (Sensor Network Encryption Protocol) that provides basic security features such as data confidentiality, two-party data authentication, integrity and data freshness. Encryption is done using the counter mode (CTR) encryption algorithm. SNEP uses CBC-MAC algorithm over the ciphered data for data authentication. It also introduces a shared counter between the sender and the receiver which is used as an initialization vector (IV) for the block cipher. The receiver and sender updates the shared counter once they have received/sent a cipher block. However, since the counter value is not being sent with the packet, there might be synchronization problems caused by dropped packets. So a re-synchronization protocol may be needed to overcome this problem.

Authors in [6] have proposed a link layer security framework known as SenSec designed for sensor networks that work on Mica2 motes. It uses a hierarchical

architecture where nodes are arranged in clusters at each level with the base station at the top of the hierarchy. It provides a resilient keying mechanism using three different keys such as global key (GK), cluster key (CK) and sensor key (SK) for combating node capture attack. Global key is generated by the base station and is present with each sensor node. Cluster key is generated by the cluster head that is assigned after the cluster formation and is used by the nodes for communication within the cluster. Every node shares the sensor key with the base station that is generated by the base station based on the id of individual sensor nodes. Although the authors have claimed that their scheme is capable of defending node capture attacks but they have not provided any explanation regarding how the different keys are being used in the various network levels for thwarting node capture attacks. Encryption has been done using a variant of Skipjack as the block cipher, called Skipjack-X. But using Skipjack-X incurs large memory usage, which is undesirable for sensor networks.

In [7] authors have proposed another link layer security solution named as SecureSense. This protocol provides security services dynamically based on runtime decisions that depend on observed external environments, internal constraints and application requirements. SecureSense is able to allocate resources optimally as it takes this decision during runtime. This decision is taken by a component known as the security broker which is inserted between the packet and radio modules. The task of the broker is to intercept the packets arriving from the radio level before passing them to the packet level and packets arriving from the packet level before passing them to the radio level. The service library invokes further cryptographic measures for fulfilling the required operations. For providing authentication SecureSense uses RC5 as the block cipher cryptographic algorithm. Though the authors have claimed that their scheme can significantly prolong network lifetime with respect to a static security model they have not dealt with any attacks that are prevalent in sensor networks.

Authors in [8] have introduced a new protocol named as MiniSec that provides security in the network layer for wireless sensor networks. The objective of MiniSec is providing confidentiality, data authentication, replay protection and data freshness. Confidentiality and data authentication in MiniSec is accomplished by applying OCB encryption [9]. Replay protection is done using a counter where the counter value is stored both at the sender and receiver ends. The sender sends few bits of the counter value with the packet. If the counter value in the packet is greater than the counter value stored by the receiver, the receiver accepts the packet else it rejects the same. MiniSec works in two modes- MiniSec-U and MiniSec-B. MiniSec-U is used in unicast communication while MiniSec-B for broadcast communication. Though MiniSec tries to minimize the energy overhead but it introduces high memory requirements in the nodes as it employs Skipjack, which requires large memory usage and also greater computation time for encryption and decryption.

In this paper an attempt has been made to design a secure link layer protocol μSec (MicroSec) for WSN, with relatively lower overhead compared to its competitors [4-8]. We have considered providing the basic security features such as confidentiality, integrity and authentication using μSec -Unicast protocol (μSec-U) as well as tried to achieve low energy consumptions keeping in mind the resource constraints of sensor nodes. The μSec-U has been developed for mica2 motes.

The rest of the paper is organized as follows. In section 2 the problem definition including the system model and the security properties considered here are described. The proposed securing scheme including its design and supporting algorithms is presented in section 3. Performance of the scheme is reported based on comparative qualitative analysis and simulation results. Concluding remarks and future scope are stated in section 5.

# 2 Problem Definition

## 2.1 System Model

The system model in the present work considers flat architecture. In this architecture the mode of communication between the nodes is unicast. We have considered mote class attackers where the adversaries are assumed to own same resources as ordinary sensor nodes. Further, we have not considered providing security services to the base station as it is highly powerful and have large amount of resources.

All the nodes are pre-deployed with a 128 bit key and a counter which is initialized to zero. Symmetric key cryptography is employed between the two nodes that communicate between themselves. The 128 bit key is used for encrypting/decrypting the message at the sender/receiver end while the counter value is used to detect the presence of replay attack.

## 2.2 Desired Security Properties

### 2.2.1 Confidentiality

Confidentiality [10] is defined as the property where the transmitted message must make sense only to the intended receiver. To all others, the message is to be considered as garbage and that will prevent other parties from discovering the plaintext. Confidentiality is typically achieved by encrypting the plaintext and sending the cipher text to the receiver.

TinySec uses CBC-encryption [11], while SNEP employs counter with CBC-MAC (CCM) encryption algorithm [12]. MiniSec provides confidentiality using OCB-encryption [9] with a non-repeating counter. We have compared several competing encryption schemes and selected the “Tiny Encryption Algorithm” (TEA) for optimum memory and CPU cycle count requirements. The justification of selecting TEA as encryption algorithm is established in section 4.1 through analysis and comparison of the other competing schemes.

### 2.2.2 Integrity and Authentication

Message integrity [13] means that data must arrive at the receiver exactly as they were sent. There must not be any change during the transmission, neither accidentally or maliciously. However, if such change has been found then the packet must be rejected by the receiver.

Authentication is a service beyond message integrity. Here the receiver needs to be sure of the sender's identity and that an imposter has not sent the message. So it empowers legitimate nodes for verifying whether a message indeed originated from another legitimate node (i.e., a node with which it shares a secret key) and was unchanged during transmission. As a result, illegitimate nodes should not be able to participate in the network, either by injecting their own messages or by modifying legitimate messages.

Typically, authentication and integrity are achieved by computing a message-authentication code (MAC). The MAC is generated by applying hash function over the payload and it is sent as part of the packet. Upon reception, the packet is considered as valid if the receiver once again computes the MAC and it matches with the received MAC.

#### 2.2.3 Replay Protection

It is an attack in which a valid data transmission is maliciously or fraudulently repeated [14]. TinySec is not resilient to such an attack. Minisec provides a counter based approach to prevent this attack. Our scheme improvises on a similar approach but it takes less memory and time requirement than that of Minisec.

## 3 Proposed Securing Scheme (μSec-U)

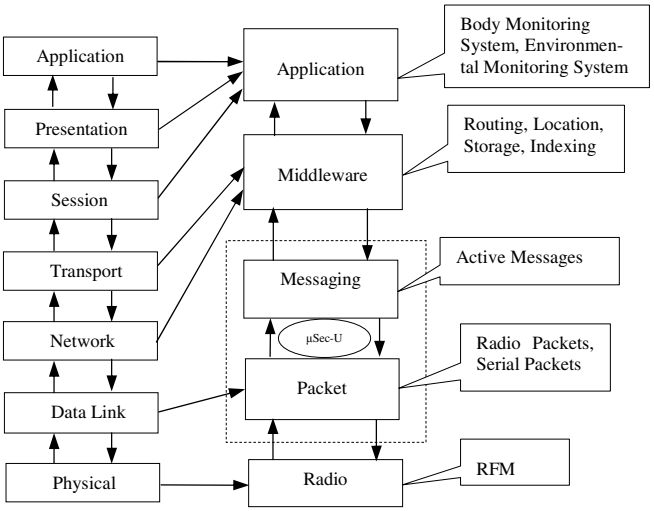
TinyOS [1] is an event driven operating system running on Mica2 motes. It does not provide any security primitives. We have developed μSec (Microsec), a protocol which not only gives basic security solutions but also prevents replay attack and enables secure, unicast communication.

### 3.1 μSec-U Design

In this section we describe the mechanism of securing unicast communication of the proposed scheme. Before describing the communication mechanism we present the protocol stack and packet format used for the scheme.

#### 3.1.1 Protocol Stack

Figure 1 shows the mapping of OSI layer protocol stack with the protocol stack of TinyOS. The functionalities [6] provided by the physical layer of the OSI model are done by the Radio layer in TinyOS. The data link layer of OSI is the packet and messaging layers in TinyOS. Similarly the middleware layer in TinyOS constitutes the network and transport layers in OSI. The work done by the session, presentation and application layers of OSI is done by the application layer in TinyOS. Generic communication interfaces is provided to the upper AM layer by TinyOS and it also uses the radio interfaces. We propose μSec-U to work in between packet and messaging layers of the TinyOS protocol stack. To be more specific μSec-U takes input from messaging layer, applies security measures and sends it for packet layer. Security services are provided in parallel to radio communication so that no additional delay is incurred.




(a) OSI layers (b) TinyOS protocol stack

**Fig. 1.** OSI vs. TinyOS protocol stack showing μSec-U

### 3.1.2 Packet Format

The packet format for TinyOS, TinySec, MiniSec and μSec-U are shown in Figure 2. The size of each attribute is denoted in bytes. The μSec-U packet is built upon the TinyOS packet with some modifications. The attributes in packet format that are shared by μSec-U with TinyOS are: Destination address, type and length.

TinyOS:

| 2 | 1 | 1 | 1 | 0…28 | 2 |
|---|---|---|---|---|---|
| Destination Address | Type | Group | Length | Payload | CRC |

TinySec:

| 2 | 1 | 1 | 1 | 2 | 0…28 | 4 |
|---|---|---|---|---|---|---|
| Destination Address | Type | Length | Initialization Vector | Source Address | Encrypted Payload | MAC |

MiniSec:

| 2 | 1 | 1 | 1 | 2 | 0…28 | 4 | 2 | 2 |
|---|---|---|---|---|---|---|---|---|
| Destination Address | Type | Length | Initialization Vector | Source Address | Encrypted Payload | MAC | FCF | DstPAN |

μSec-U:

| 2 | 1 | 1 | 2 | 0…28 | 1 | 4 |
|---|---|---|---|---|---|---|
| Destination Address | Type | Length | Source Address | Encrypted Payload | Counter | MDC |

**Fig. 2.** Packet formats

The attribute length is used for specifying the length of the payload. The attribute 'type', however, is used for future extension of our scheme for securing broadcast communication. The attribute that is changed from TinyOS is the CRC field. The CRC field is replaced by MDC (Modification Detection Code) while an additional 1 byte counter field is added to the μSec-U packet. The attribute counter is used for replay attack detection and the attribute MDC is used for checking message integrity.

### 3.1.3 Encryption

We have modified Tiny Encryption Algorithm (TEA) and proposed modified TEA (mTEA). The μSec-U uses mTEA algorithm as the block cipher that performs necessary encryption and decryption on each of the 64 bit blocks of the payload at the sender/receiver end respectively.

*Brief on TEA:* Tiny Encryption Algorithm (TEA) [15] has 64 rounds of simple arithmetic and logic processes comprising of shifts, additions and XORs. It uses 128-bit key (K) and 64-bit block size. The 128-bit key (K) is split into four 32-bit blocks or sub-keys K[0], K[1], K[2], K[3]. The sub-keys K[0] & K[1] are used in one of the rounds (odd round) and K[2] & K[3] are used in the other round (even round). The inputs to the encryption algorithm are a 2-tuple (plaintext block, key) data. Each block of data is divided into two halves. Encryption is performed on these two halves of each data block for 64 rounds by both left and right shifting of bits causing all bits of data as well as key to be mixed repeatedly. After the completion of 64 rounds, the encrypted cipher text is generated and is sent to the receiver. Similarly at the receiver end, decryption is performed that involves the same procedure as in encryption with the difference being only in the use of the sub-keys in reverse order.

The function part of TEA algorithm that comprises of shifting, addition and xoring operations is denoted as-

$$f(M, K[j, k], \partial[i])=((M<<4)+K[j]) \oplus (M+\partial[i]) \oplus ((M>>5)+K[k])$$

where M is the message, K[j, k] denotes the $j^{th}$ and $k^{th}$ sub-keys, i is the round number and δ is the golden number [14] defined as- $\left(\sqrt{5}-1\right)2^{31}$.

*Proposed modification of TEA (mTEA):* The modification of the TEA algorithm has been done in the function part by including number of rounds (N) as part of the function. Further, 64 rounds involve enormous computations that are neither desirable for resource constrained sensor nodes nor essential for most of the WSN applications that require short lifetime. So security level has been made adaptable by changing N as per the need. The function part of mTEA is denoted as-

$$f(M, K[j, k], \partial[i])=((M<<4)+K[j]) \oplus (M+\partial[i]) \oplus ((M>>5)+K[k])<<N$$

### 3.1.4 Hashing

Hashing has been done for providing authentication and integrity to the message being sent from the sender to the receiver. The process of hashing has been implemented in μSec-U in two steps:

- Use compression function to produce fixed length compressed outputs
- Generate Modification Detection Code (MDC) or unkeyed hash function from Substitution box (S-box)

**Table 1.** Substitution box

| I/P Range of Compression Function | Hash Value |
|---|---|
| Value : 0x00 – 0x1f | Value \| Value \| Value \| Value |
| Value : 0x20 – 0x3f | Value \| ~Value \| Value \| ~Value |
| Value : 0x40 – 0x5f | ~Value \| Value \| ~Value \| Value |
| Value : 0x60 – 0x7f | Value \| Value \| ~Value \| ~Value |
| Value : 0x80 – 0x9f | ~Value \| ~Value \| Value \| Value |
| Value : 0xa0 – 0xbf | Value \| Value \| Value \| ~Value |
| Value : 0xc0 – 0xdf | ~Value \| Value \| Value \| Value |
| Value : 0xe0 – 0xff | ~Value \| ~Value \| ~Value \| ~Value |

The first step of hashing involves the compression operation where the inputs are the least significant 8 bits of the encrypted output from each block. Each of the inputs is compressed to produce 2 bits as output and stored in a temporary buffer. The compression is done using modulo 4 operation. After all the blocks undergo compression operation, the results are appended to form the output comprising of 2t bits considering t data blocks. Once the compression is done, in the next step, MDC (Fig 4: μSec-U packet format) is generated and is transmitted along with the encrypted payload. To generate MDC, concept of S-box [16] has been used as it obscures the relationship between the key and the ciphertext. The output of 2t bits obtained after compression is considered as inputs to the S-box (Table 1: Substitution box). The corresponding input is matched with the inputs in the S-box to obtain the output consisting of 4 bytes.

### 3.2 Transmission

Once the encryption and hashing are completed, the encrypted payload and MDC are ready for preparing the packet for transmission except the value of counter. All nodes maintain individual counter values for each of its neighbouring nodes. To start with, the counter values maintained by all the nodes are initialized to zero. When a node (sender) starts transmission with another node (receiver), the sender increases its counter (corresponding to the receiver node) value by one. Similar to MiniSec-U, it also uses the concept of LB (Last Bits) optimization where instead of sending the entire counter bits (the size of the counter is 4 bits) only the last 8 bits of the counter are sent with each packet. After completion of 8 rounds the hash result of all blocks, encryption result and the LB bit of the counter are appended together to form the 7-tuple packet containing destination address, type, length, source address, encrypted payload, counter and MDC which is to be sent to the receiver/ destination.

### 3.3 Reception

At the receiving end, the receiver checks the destination address (μSec-U packet: Figure 2) of the received packet for verifying whether the packet is destined for the receiver.

#### 3.3.1 Replay Attack Detection

The receiver compares the counter value in the 7-tuple packet with the counter value stored (section 3.2) by it. If both the counter values match then it can be confirmed that no replay attack has taken place and the packet is accepted. Once the packet is accepted the receiver increments the counter value by one.

#### 3.3.2 Confidentiality Check

If there is no replay attack, the payload is split into blocks each of 64 bits and on each block mTEA algorithm is run for decrypting the data in each block for 8 rounds. If it is successfully decrypted, confidentiality is ensured.

#### 3.3.3 Integrity Check

Once the blocks are decrypted the hash function is employed on each of the 64 bit decrypted blocks. The hash results of all the blocks are concatenated and if this hash result matches with MDC of the received packet then integrity is preserved.

### 3.4 Algorithm for μSec-U

The μSec-U algorithm for unicast communication is presented in two parts- one is run in a sender node and the other in a receiver node.

#### 3.4.1 Sender

The following algorithm is running at sender side.

**Input:** Message X to be transmitted from node s to a node r.
**Output:** 7-tuple packet ready for transmission

**Begin**

1. apply unambiguous padding on X
2. obtain X' // formatted message- $X' = X \| 1 \| 0^j$, where j is the smallest positive integer such that $L + j \equiv -1 \pmod{64}$
3. split X' into t blocks each of 64 bits // $X' = x_0 x_1 x_2 ... x_{t-1}$
4. initialize H, E(X') to $\phi$ // H and E(X') store the hash value and encrypted m sage respectively
5. **for** (i=0; i≤t-1; i++)
6. **for** (j=0; j≤7; j++) // 8-round encryption start
7. run mTEA on each block $X_i'^j$ // for ensuring confidentiality
8. **end for**
9. obtain $E(X'_i)$ // obtain encrypted $i^{th}$ block of X'
10. compute $E(X')=E(X') \| E(X'_i)$ // form encrypted payload
11. compute $H_i=$(Extract LSB $E(X'_i)$) mod 4

12. compute $H=H \| H_i$
13. **end for**
14. search S-box to find out the matched value for H
15. obtain search_result
16. compute MDC=search_result
17. generate 7-tuple packet // attributes of the packet are destination address, type, length, source address, E(X'), $counter_s^r$ , MDC
18. increase value of $counter_s^r$ by one
19. send 7-tuple packet to receiver

**End**

### 3.4.2 Receiver

**Input:** 7-tuple packet from sender/transmitter node
**Output:** Detection of replay attack and checking of basic securities

**Begin**

1. receive 7-tuple packet from sender
2. check packet header i.e. destination address // verify whether the packet is for the intended receiver
3. retrieve $counter_s^r$ value from the packet
4. **if** ( $counter_s^r$ value in the packet = $counter_s^r$ value stored in the receiver) // check for replay attack
5. accept packet
6. increase $counter_s^r$ value by one
7. **else**
8. reject packet
9. **exit**
10. obtain encrypted payload $E(X')$ from the packet
11. split the payload $E(X')$ into t blocks each of 64 bits // $X' = x_0 x_1 x_2 ... x_{t-1}$
12. initialize D(X') and H to $\phi$ // H and D(X') are to store the hash value and decrypted message respectively
13. **for** (i=0; i≤t-1; i++)
14. **for** (j=0; j≤7; j++)
15. run mTEA on each block $X_i'^j$ // for confidentiality check
16. **end for**
17. obtain $D(X'_i)$ // obtain decrypted $i^{th}$ block of X'
18. compute $H_i$ =( Extract LSB $D(X'_i)$) mod 4
19. compute H=H || Hi
20. **end for**
21. search S-box to find out the matched value for H
22. obtain search_result
23. compute MDC = search_result
24. **if** (computed MDC = MDC in the packet) // integrity and authenticity check

25. accept packet
26. **else**
27. reject packet
28. **end if**
**End**

# 4 Performance Analysis

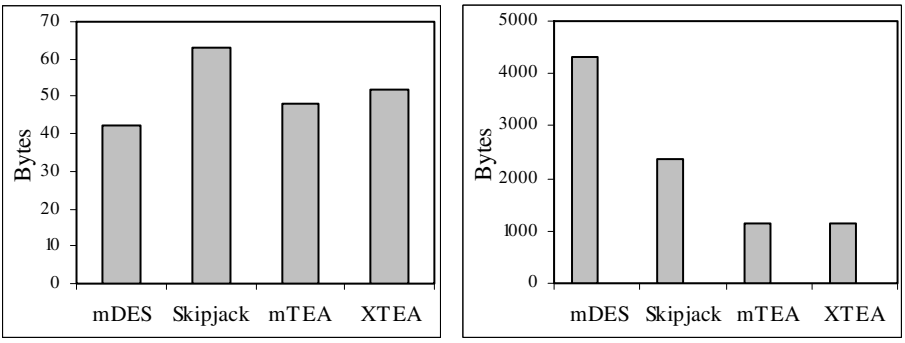
The effectiveness of the proposed scheme for securing unicast communication reported in the earlier section is evaluated both by qualitative and quantitative analyses.

## 4.1 Qualitative Analysis

In this section primarily justification of selecting mTEA over other existing algorithms for encryption/ decryption is provided through qualitative analysis based on their memory requirements, CPU cycle count and average throughput to code size ratio.

### 4.1.1 mTEA vs. Its Competitors

*Memory requirements*
The cryptographic algorithm mTEA used in μSec-U has been compared with other existing cryptographic algorithms such as modified DES (mDES) [10], Skipjack [8], XTEA [10]. The memory usages in Figure 3 clearly reflect that mTEA outperforms all the other cryptographic algorithms with respect to total memory requirements.




(a) RAM (b) ROM

**Fig. 3.** Memory requirements for different ciphering schemes

*CPU cycle count*
The CPU cycle time requirement gives a measure of computational complexity and energy consumption. The amount of energy consumed is directly related to the CPU cycle time. Considering the amount of energy consumed per CPU cycle is fixed,

energy consumption is measured in terms of number of CPU cycles required for executing per data block of plaintext/ciphertext processed. The CPU cycle time required for encryption and decryption of various algorithms is shown in Figure 4. We observe from the figure that encryption and decryption cycle counts are minimum for mTEA thus ensuring that energy consumed is also least.

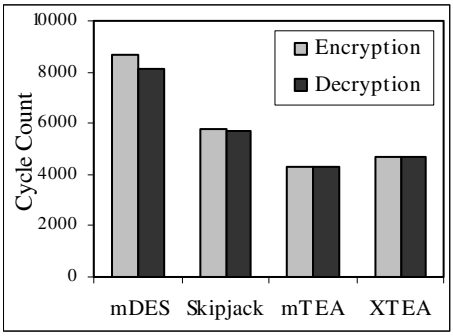



**Fig. 4.** Cycle Count for encryption and decryption

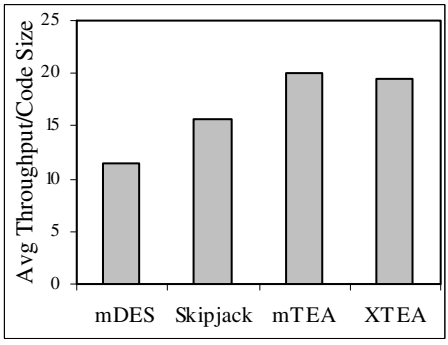



**Fig. 5.** Ratio of Average Throughput (bits/sec) to Code Size

*Average throughput to code size ratio*

We have also used the ratio of average throughput to the code size in logarithmic scale as a parameter for comparison. The ratio is calculated as-

$$10\times\log_{10}\left(\frac{\text{Average Throughput}}{\text{Code Size}}\right).$$

As throughput is defined as the number of bits processed per second by an algorithm, it influences the performance of the algorithm. The ratio described above can be regarded as the figure of merit for the encryption/decryption algorithm. The value of this ratio for the different ciphering schemes is shown in Figure 5.

### 4.1.2 Overhead Comparison

Table 2 gives a comparison of μSec-U with TinyOS, TinySec and TinySec (AE). The comparison table illustrates that μSec-U outperforms all the other protocols in terms of security overhead. As all the schemes have evolved from the basic TinyOS scheme, the last two columns in the table are the illustration of how much increase in overhead over TinyOS has taken place in all the schemes with respect to packet size and bandwidth respectively. In packet overhead calculation we have used the term byte time, which is the time taken to transmit a single byte of data over the radio [4]. Also a 28-byte start symbol and a 4-byte synchronization message are also introduced as an extra overhead.

**Table 2.** Comparison of packet overhead

| Scheme | Payload size (Bytes) | Packet overhead (Bytes) | | Total Size (Bytes) | Transmission time (ms) | Size increasing over TinyOS (%) | BW overhead over TinyOS (%) |
|---|---|---|---|---|---|---|---|
| | | Header overhead | Security overhead | | | | |
| TinyOS | 24 | 39 | - | 63 | 26.2 | - | - |
| TinySec | 24 | 38 | 2 | 64 | 26.7 | 1.58 | 1.7 |
| TinySec(AE) | 24 | 40 | 4 | 68 | 28.8 | 7.9 | 7.3 |
| SenSec | 24 | 40 | 4 | 68 | 28.8(e) | 7.9 | 7.3 |
| Minisec | 24 | 47 | 3 | 74 | 31.3(e) | 17.38 | 16.9 |
| μSec-U | 24 | 40 | 4 | 68 | 28.8(e) | 7.9 | 7.3 |

## 4.2 Quantitative Analysis

In this section simulation results are presented.

### 4.2.1 Simulation Environment

Simulation of μSec-U is performed using TOSSIM (version 1.0), which is a discrete event simulator for WSN having nodes supporting TinyOS. In simulation the key has been randomly assigned, with every node having a unique key. We consider radio model as "simple" which is implemented in TOSSIM for testing single-hop algorithms. TOSSIM does not provide any power/energy consumption model. We assume the random model of ADC [17] where whenever sampling takes place it returns a 10-bit random value. Further, TOSSIM models EEPROM by means of a large, memory mapped file which is anonymous and is not available at the end of current session. However, we have used a name file which allows the EEPROM data to persist across multiple simulation invocations.

**Table 3.** Simulation Parameters and values

| Parameters | Value |
|---|---|
| Number of nodes | 5 |
| Radio propagation speed | $3\times10^8$ meters/s |
| Radio speed | 19.2 Kbps |
| Payload size | 24 bytes |
| CPU clock speed | 8 MHz |

We implemented μSec-U in nesC code [18], the programming language used for TinyOS. The relevant parameters used for simulation and their associated values are listed in Table 3.

### 4.2.2 Simulation Metric

As energy consumption by a node is greatly influenced by the time taken for cryptographic computations, we have analyzed the cryptographic computation time in a node. The cryptographic computation time considers the time required for encryption /decryption at the sender/receiver.

### 4.2.3 Simulation Results

We have simulated our proposed μSec-U scheme using TOSSIM for determining the efficiency of the ciphering algorithm employed in the scheme as well as for evaluating the authentication/integrity and replay attack protection mechanism used in the same. For these experiments, a network consisting of 5 sensor nodes are used for accurately determining the details of communication. The simulation is performed at a scale of 0.5 times the real speed in order to study the message exchange between the nodes. We have run the simulation 10 times and each time results are collected from one randomly chosen node. Finally, we have taken the average of the results and plotted.

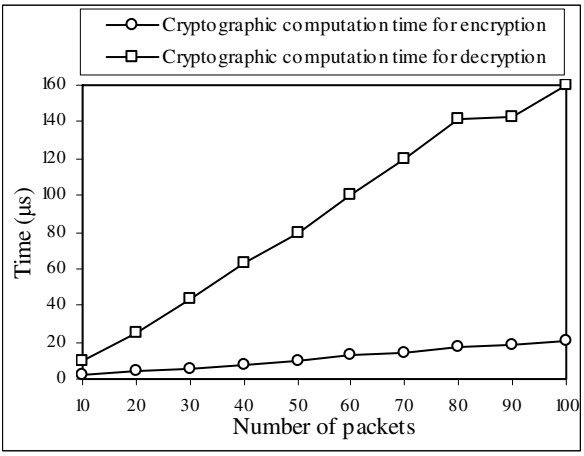


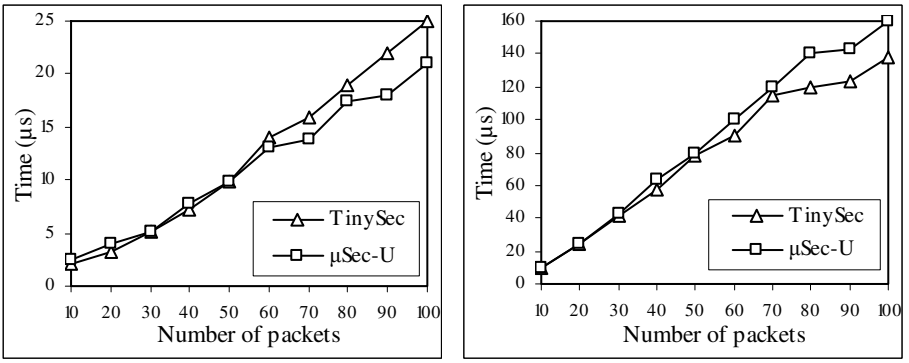

**Fig. 6.** Cryptographic computation time

Time required for cryptographic computations for encryption and decryption are plotted in Figure 6. Our primary observation is that as the number of packets increases the cryptographic computation time required for both encryption and decryption increases. To be more specific if number of packets is 10, cryptographic computation time required for encryption is 2.5 μs whereas for decryption it is 10 μs. Also if number of packet is 100, cryptographic computation time required for encryption is 21 μs and same for decryption is 160 μs. This is due to the fact that as the number of packets increases the time required for cryptographic computation also increases. Also it is observed that cryptographic computation time required for decryption is more than that for encryption. This is because during decryption more number of constraints are involved such as checking the packet header and retrieving the counter value from the received packet before doing the actual decryption process.

### 4.2.4 Comparative Study

Further, we compare μSec-U with TinySec based on the parameter cryptographic computation time for encryption and decryption.

Figure 7(a) shows that with number of packets up to 50, time required for encryption of the packet in μSec-U is nearly same as that of TinySec. But when the number of packets increases beyond 50, the time required in μSec-U is lesser than TinySec by about 18%. However, we observe that computation time required for decryption (Figure 7(b)) is about 11% more in case of μSec-U than TinySec. This is due to the constraints involved in counter value checking at the time of reception. This small overhead is affordable for getting replay attack protection which is not provided by TinySec.




(a) for encryption (b) for decryption

**Fig. 7.** Time required for cryptographic computations

## 5 Conclusion and Future Works

The proposed securing protocol μSec-U offers defense against replay attack in addition to provide fundamental security features for unicast communication. During the protocol design emphasis is given such that it meets both the desired requirements of keeping it lightweight in presence of resource-constrained nodes while achieving high-level security features. It uses lightweight cryptographic algorithm requiring less

overhead in terms of computations and energy consumption. The μSec-U has been simulated in TOSSIM simulator, the simulator used for simulating protocols based on TinyOS. A detailed comparative analysis is performed where it is shown that our scheme requires significantly less overhead than its competitor scheme while both the schemes defend replay attack in addition to providing basic securities. To make our solution a complete one, as a future extension, the scheme may be extended for securing broadcast communication.

## References


1. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., Pister, K.S.J.: System Architecture Directions for Networked Sensors. ACM SIGPLAN Notice 35(11), 93–104 (2000)
2. Perrig, A., Stankovic, J., Wagner, D.: Security in Wireless Sensor Setworks. Communications of the ACM, Special Issue on Wireless Sensor Networks 47(6), 53–57 (2004)
3. Ghosal, A., Halder, S., DasBit, S.: A Dynamic TDMA Based Scheme for Securing Query Processing in WSN. Wireless Networks 18(2), 165–184 (2012)
4. Karlof, C., Sastry, N., Wagner, D.: TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. In: Proc. of 2nd Int'l Conf. SenSys, pp. 162–175 (2004)
5. Perrig, A., Szewczyk, R., Wen, V., Culler, D., Tygar, J.D.: SPINS: Security Protocols for Sensor Networks. In: Proc. of 7th Int'l Conf. MobiCom, pp. 189–199 (2001)
6. Li, T., Wu, H. Wang, X., Bao, F.: SenSec Design. $I^2$R Sensor Network Flagship Project (SNFP: Security part). Technical Report-TR v1.0 (February 2005)
7. Xue, Q., Ganz, A.: Runtime Security Composition for Sensor Networks (SecureSense). In: Proc. of IEEE VTC, vol. 5, pp. 2976–2980 (2003)
8. Luk, M., Mezzour, G., Perrig, A., Gligor, V.: MiniSec: A Secure Sensor Network Communication Architecture. In: Proc. of 6th Int'l Conf. IPSN, pp. 479–488 (2007)
9. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. In: Proc. of ACM Int'l Conf. CCS, pp. 96–205 (2001)
10. Brands, S.A.: Rethinking Public Key Infrastructures and Digital Certificates Building in Privacy. MIT Press (2000)
11. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. In: Proc. of 38th Annual Symposium FOCS, pp. 394–403 (1997)
12. Housley, R., Whiting, D., Ferguson, N.: Counter with CBC-MAC (CCM), Submitted to N.I.S.T. (June 3, 2002), `http://csrc.nist.gov/encryption/modes/proposedmodes/`
13. Menezes, A., Oorschot, P.V., Vanstone, S.: Handbook of Applied Cryptography. CRC Press (1996)
14. David, R.R., Marchany, R.C., Midkiff, S.F.: Scalable, Cluster-based Anti-replay Protection for Wireless Sensor Networks. In: Proc. of IEEE Workshop on Information Assurance, pp. 127–134 (2007)
15. Wheeler, D., Needham, R.: TEA, A Tiny Encryption Algorithm. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 363–366. Springer, Heidelberg (1995)
16. Murphy, S., Robshaw, M.: Essential Algebraic Structure within the AES. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 1–16. Springer, Heidelberg (2002)
17. Levis, P., Lee, N.: TOSSIM: A Simulator for TinyOS Networks. User's manual, in TinyOS Documentation (March 23, 2007)
18. Gay, D., Levis, P., Behren, R.V., Welsh, M., Brewer, E., Culler, D.: The nesC Language: A Holistic Approach to Network Embedded Systems. In: Proc. of Programming Language Design and Implementation, pp. 1–11 (2003)