Implementing the TEA algorithm on Sensors

Shuang Liu
Department of Computer Science
The University of Alabama
Box 870290
Tuscaloosa, AL 35487-0290
sliu@cs.ua.edu

Olga V. Gavrylyako
Department of Computer Science
The University of Alabama
Box 870290
Tuscaloosa, AL 35487-0290
ogavrylyako@cs.ua.edu

Phillip G. Bradford
Department of Computer Science
The University of Alabama
Box 870290
Tuscaloosa, AL 35487-0290
pgb@cs.ua.edu

ABSTRACT

Sensors are tiny computers with limited computational capability and physical resources. The implementation of secure protocols for sensor network is a big challenge. In order to provide high security for sensor networks, it is very important to choose a small, efficient and effective encryption algorithm as a security primitive. The TEA (Tiny Encryption Algorithm) is an efficient algorithm that requires little memory and resources. These features make the TEA a good candidate for security mechanism for sensors.

In this paper we describe an implementation of the TEA algorithm on the platform of sensor networks (Berkeley Motes). In our experiment, the data packets obtained from photo and temperature sensors are encrypted on the sensor node using the TEA algorithm. After that, they are sent to the base station by radio. The base station will receive the data packets and forward them to attached PC, where the data packets are decrypted and displayed. We also propose a particular approach to efficiently evaluate the performance of the TEA in terms of execution time on sensor nodes.

Keywords

Sensor network, Motes, TinyOS, TEA, Security

1. INTRODUCTION

Recent progresses in wireless networking technology and micro-electro-mechanical systems lead to the emergence of a new product in the post-PC era - networked sensors. Sensor networks with ad-hoc configurations and programable sensors will be widely used in a variety of scenarios. Futurists even envision that thousands of ultra-tiny sensors would be embedded into environment using the environmental energy, assisting people to acquire more physical information [2].

The unusual applications of sensor networks require each sensor to be highly miniaturized in terms of physical size, power consumption and individual price. This will present some design challenges that the intelligent applications are to be built on the platform of the limited amount of memory, the shortage of computing power and a stringent radio transmission bandwidth.

ACMSE '04, April 2-3, 2004, Huntsville, Alabama USA

However, it is not surprising that security is required in many applications when transmitting sensitive data over the vulnerable broadcasting sensor network. A third party can receive and tamper raw message packets if there is no secure mechanism to protect the transition. Several protocols related to authentication, confidentiality and secure routing were presented in [5, 7]. Generally, all the security strategies are based on the underling cryptographic primitives. To implement these security protocols on the resource-constrained sensors, appropriately choosing efficient encryption algorithms is a major issue.

2. SECURITY IN SENSOR NETWORK

In the sensor network, sensors are organized into the specific configuration to satisfy the requirements of ad-hoc applications. Unfortunately, the connectivity can not remain stable at any working time. And the sensor network is a broadcast network, in which any signal can be captured by adversaries. These features make wireless ad-hoc sensor networks more vulnerable than wired networks. This presents real challenges in the implementation of the following security requirements [5].

Data confidentiality: The transmission of sensitive data should be protected for both sender and receiver in the sensor network. The typical approach to achieve this is to encrypt data using keys that are held by the receiver and sender.

Message authentication: This mechanism can ensure that received data is really sent by a claimed sensor. Without authentication, malicious data can be injected, destroying the whole network.

Data integrity: Due to the broadcast medium in sensor networks the message packet might be tampered by an adversary during the transmission. Some mechanisms must be provided to detect altered packets and protect data integrity.

Due to inherent constraints on sensor node, there is a trade-off between security and the consumption of resources. If an application is designed to protect data more securely, the higher overhead has to be paid in the computing and communicating resources.

The majority of existing algorithms cannot be implemented to secure the system. During the running time, the overhead caused by complex encryption algorithms would greatly influence the performance of other processes in the operation system. On such a capability-restrictive and overloaded platform [6], only a small fraction of computing and memory space can be devoted to cryptographic algorithms. Thus, some inexpensive encryption algorithms are the candidates for being implemented on sensors.

In [4], the authors surveyed several inexpensive encryption

algorithms based on simplicity and efficiency. Those algorithms include both symmetric and asymmetric encryption algorithms. In asymmetric cryptography, the cost of generating public and private keys is relatively high. It is not so suited to the resource-constrained sensor network. Symmetric cryptography is much cheaper, so it may suffice for the sensor network with physical security [4]. In the sensor network that lacks physical security, the mechanism for the key management could supplement the shortage of key compromise inherently in the symmetric cryptographic [5].

3. THE TEA ALOGRITHM

British researchers from Cambridge University proposed extremely simple encryption algorithm, the TEA (Tiny Encryption Algorithm) [8]. It based on an alternative application of a large number of iterations with XORs and additions, rather than on preset tables. So it can achieve better performance with smaller code size and less complexity than standard encryption algorithms[3]. The author claimed that the TEA is three times faster than popular encryption algorithms such as DES.

The TEA encrypts 64 data bits at a time using a 128-bit key, which is the strongest encryption so far. The TEA uses a 128-bit master key *K[0..3]* and derived subkeys. The Key schedule is simple. Odd rounds use *K[0,1]* as the round subkey, and even rounds use *K[2,3]*. Originally designed for 64 bit plaintext blocks, later the TEA was extended for larger block sizes, in which the iterative times and key scheduling are slightly changed.

In SPINS [5] the authors implemented encryption on resource-constrained devices using RC5 encryption algorithm. However, the TEA is much more efficient in terms of static size and running time consumption. Although the result of thorough scrutiny of the TEA has been unknown in cryptanalysts, we still explore the applicability of the TEA in our experiment. Its extra-ordinary simplicity and efficiency are very important features for potentially miniaturized hardware. Moreover, the size of message packets in specific sensors is often fixed, so it is more acceptable to view each message packet as data chunk and use the Block TEA algorithm to encrypt the packet.

4. MOTES HARDWARE AND TINYOS

The sensor network is a novel research field appearing recently. Its unusual requirements present unconventional design issues in the hardware infrastructure and software system. In the following section, these problems are briefly discussed.

4.1 Mote Hardware

Sensors used in this project are based on *ATMEGA128L* micro-controller [1]. It is slightly larger and more powerful than sensors in real applications, but they have the commonly important characteristics representing real networked sensors. In more details, experimental sensor Mote has following components:

- The processor is the ATMEL 90LS8535 (4 MHz), which is a 4MHz Harvard architecture with 8-bit addresses. It has 128-Kbyte program flash memory, 4KB SRAM. It operates at 4 MHz and 3.0V, and contains internal timers/counters. The 90LS8535 contains a UART controller that is connected to a base station host computer's serial port. It has three sleep modes: idle, power down, and power save, which can reduce the energy consumption at the idle time of processor.
- Three LEDs can output analog signals through I/O port. They may be used as an approach of debugging.
- Radio component consists of RF Monolithics 916.50 MHz transceiver, antenna and a collection of physically configuring components. The transmission distance is 500 ft.
- Coprocessor
- Sensor board provides temperature sensors, light sensors, thermometer and microphone.

4.2 TinyOS Platform

TinyOS is an event-driven operating system designed specifically for Motes [1, 5]. Like conventional operating systems, TinyOS seeks to provide abstractions of physical devices and implementation of common functions to reduce the burden of application development. However, due to the challenges related to sensors, TinyOS has its own exceptional features.

4.2.1 Software Architecture

TinyOS deals with novel requirements. The system is required to be able to handle information flows fluently on a hardware platform without sufficient buffering. So TinyOS was presented to manage the hardware capabilities effectively, and support concurrency-intensive operations in a manner that it can achieve efficient modularity and robustness. The main focus of TinyOS is to provide a rich expression of concurrency within limited resources. TinyOS has chosen an event-based model that was borrowed from parallel and distributed computing systems. This model provides an advantage that high levels of concurrency can be handled in a very small amount of space by allowing independent components to share a single execution context. A complete TinyOS consists of a tiny scheduler and a graph of components. The tiny scheduler uses simple FIFO strategy and it is also power-aware. It can dynamically shut down and wake up the whole system when the number of tasks in the task queue is changed. A number of hierarchical components construct the whole infrastructure of TinyOS. The physical hardware represents the lowest level of components. Within each component, there are four interrelated parts:

- A set of command handlers (synchronous function call)
- A set of event handlers (asynchronous function call)
- A fixed-size frame (memory)
- A bundle of tasks (executable programs)

Each component declares commands it issues or receives and events or signals it handles. The components in the system are able to interact with each other through interfaces. Higher-level components issue commands to lower level components and lower level components signal events to the higher-level components. So for a programmer, it is easy to connect high-level components with TinyOS by simply following the interface standards required by TinyOS.

4.2.2 Message Processing Sequence

In a sensor network, a number of sensor nodes are tied together by radio connection. Each sensor has a set of system components, which are able to collaborate with each other to perform a particular task. Figure 1 shows the typical operational chain of internal components in TinyOS. Figure 1 is based on a figure in [2].

A timer event initiates the sensing application component to start periodical data collection. When data is collected,

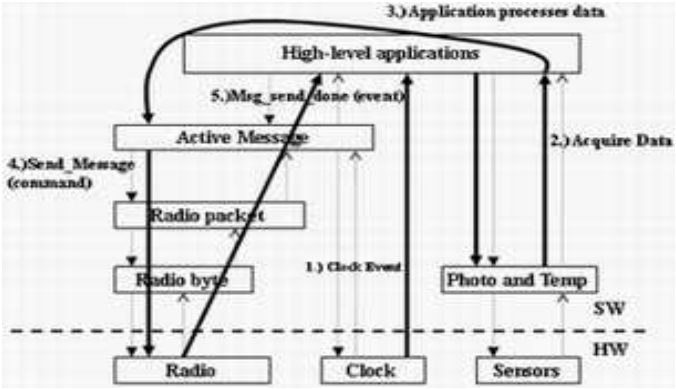

Figure 1: Message Processing Sequence in TinyOS.

the **send_message** command is issued to initiate the transfer. This command produces a chain of commands and events in the underlying components. When message is successfully broadcast over radio, the Active Message component sends the **msg_send_done** event to inform the high-level application component that the transmission task is done.

5. IMPLEMENTATION AND EVALUATION

On the hardware and software platform described in Section 5.1 and 5.2, we implemented the TEA algorithm. Based on this simple and basic security application in our experiment, more sophisticated security strategies and protocols can be built.

5.1 Experiment Description

The main goal of our experiment is to investigate the applicability of the TEA implementation on Motes and to measure its running time. We basically focus on the aspect of implementation, so there are no complicated security protocols in our application. We also eliminated routing functionality.

Therefore, the infrastructure of sensor network in our experiment is straightforward. There is only one sensor node and one base station forming a simple sensor network. The base station is connecting with a host PC through serial port. We should guarantee that the sensor is always in the radio range of the base station, so that loss rate of packets should be maximally reduced. For the simplicity of the implementation, the initial key of the TEA is a fixed 128-bit number, which is given initially, and never changed during the experiment.

We implement the TEA algorithm on the sensor node, and encrypt data collected from photo and temperature sensors. Then, the encrypted data are encapsulated into packets and sent to the base station. Upon receiving a packet, the base station directly forwards the packet to attached PC, where the decryption algorithm is running to decode packets and display it onto the screen.

In order to measure how much time the TEA spends to encrypt message on TinyOS we propose a special approach. Just before the TEA algorithm begins to run on the sensor node, we record the current system time *T1*, and start the algorithm. When the execution of the algorithm gets closed, we record another system time. Obviously, the difference of two time records will be the running time of the TEA algorithm. So the simplest way to accomplish this is to timestamp the two messages that are immediately prior and posterior to the execution of the TEA, and calculate the time interval for the running time of the TEA. However, there are some practical issues we have to deal with.

Considering the lack of the precise timer in a sensor and concurrency-intensive nature of the sensor network, technically it is very hard to precisely control the clock in a sensor. The sensor would probably give the wrong time stamp due to the processing delay. So the approach using time stamp in a sensor is not feasible in our experiment. However, the clock in the host PC is more precise and it is also more powerful, so the timing process should be set up on the base station.

In our approach, after a sensor finishes collecting data, the sender sends out a beginning synchronous signal to the base station. This signal is used to inform base station to start the timer. Similarly, we have to use other type of message to end timing. But, since we broadcast the encrypted data immediately after the time that encrypting process ends, the data packet is a good signal to stop timer on the host PC.

$T_{start}^{encrypt}$: Starting time on the first sensor with TEA.
$T_{end}^{encrypt}$: Stopping time on the first sensor with TEA.

$$\Delta T_1 = T_{start}^{encrypt} - T_{end}^{encrypt} \quad (1)$$

However, ΔT_1 is not a precise time span that sensor spends on the TEA. Some system processing overhead, such as interrupting process and transmission procedure would be added to ΔT_1, thus increasing the value ΔT_1. In order to get more accurate time span, we need a new method to offset the time span taken by the system process. So we introduce the second identical sensor node that performs the same functionality as the first sensor, but we do not have the TEA algorithm running on the second sensor node.

T_{start}: Starting time on the second sensor without TEA.
T_{stop}: Stopping time on the second sensor without TEA.

$$\Delta T_2 = T_{start} - T_{end} \quad (2)$$

Since all experimental platforms are the same, the system overhead for the second sensor node should be same as the overhead for the first one. So the time span that the TEA algorithm spends on TinyOS can be calculated by the following formula:

$$\Delta T_{TEA} = \Delta T_1 - \Delta T_2 \quad (3)$$

According to above experimental description, there are three types of software we should develop:

- The software on the sensor node. It is a high-level application on TinyOS written in NesC. It operates with Motes hardware to perform the functionalities of sensing data, sending data and encrypting data.
- The software on the base station. It collects packets from the sensor node over radio and forward the raw data packets directly to the PC machine attached through serial port. It is also a TinyOS program.
- The software running on host PC, which is responsible for displaying received data on the screen and computing the time span of the TEA running time.

5.2 Message Packet Specification

Our experiment uses two types of message packets. Each time when we want to start measuring the running time of the TEA, the sensor node will send a synchronous message packet to the base station. It is only used as a starting

Dest Addr	Handler ID	Group ID	Msg len	Photo data	Temp data	Blank block
2 bytes	1 byte	1 byte	1 byte	2 bytes	2 bytes	24 bytes

Message header — Data Payload

Figure 2: The format of the data message packet

Dest Addr	Handler ID	Group ID	Msg len	Counter
2 bytes	1 byte	1 byte	1 byte	4 bytes

Message header — Data Payload

Figure 3: The format of synchronous message packet

signal, carrying no other data. The Data message packet immediately follows the synchronous message packet. The data message packet contains the temperature and photo data encrypted using the TEA algorithm.

Generally, the message packets in our experiment are simple, because we only test the performance of the TEA running on sensor node, instead of transmitting complicated information. Figure 2 represents the format of data message packet. The message packet used in our experiment is similar to the original message packet of TinyOS, which has five-byte header, payload of 29 bytes and two-byte CRC. We do not change the header of the TinyOS packet format. Due to convenience of applying the Block TEA encryption, we reduce the length of payload to 28 bytes and do not use CRC bytes. Additionally, in the data payload part, we use 4 bytes for the photo data and temperature data sensed from underlying hardware sensors. The rest of the space in the data payload is filled with 0. The format of synchronous message packet is illustrated in Figure 3. Similarly, the first five bytes are the header of the packet. In order to control the sequence number of the packet, we add 4 bytes in data payload for the counter information. The counter is incremented by one for each time the synchronous message packet is broadcast.

5.3 Sensor Software Implementation

Programming on TinyOS is component-based. The only thing we need to implement is to write our own components and wire them with the existing components that can perform some functionality using the specific protocol. In the software of sensor node, there are six components we will use:

- *Main* Component. It is a system component dealing with the basic control of the whole sensor, such as initializing a sensor node and starting processing.
- *FairyM* component. This component is implemented by us. It performs main functions such as collecting data, encrypting data, and sending data. It lies in the highest level of the OS.
- *Comm* Component. All radio communication is handled by this system component, which can issue sending commands and receiving returning events to notify the upper-level components that the sending procedure is done.
- *PhotoTemp* Component. This system component interacts with the underlying hardware of photo and temperature sensor, so that the Photo and temperature data can be appropriately collected into the buffer.
- *LedsC* Component. In order to display the working state of a sensor, this component can issue command to control Leds on the sensor board
- *TimerC* Component. Since all processing work begins from the timer event, the TimerC system component is mainly responsible for handling timer events.

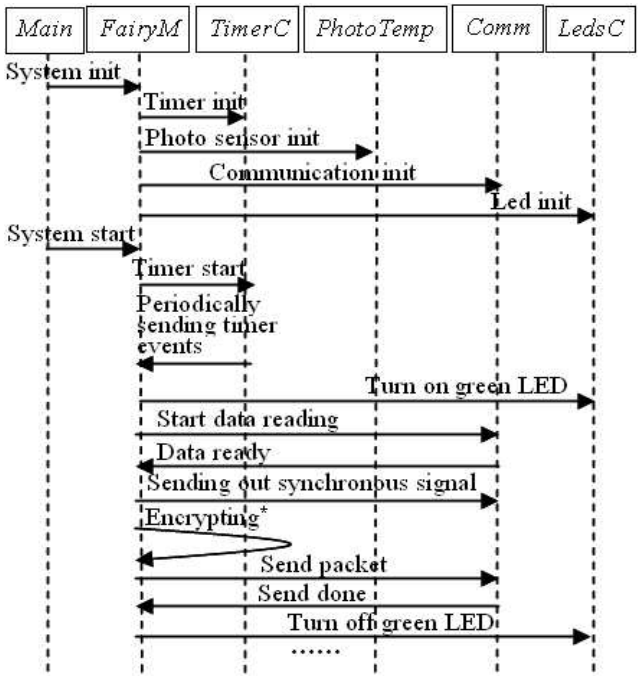

Figure 4: Sequence of interaction in components of sensor.

Figure 4[1] is the interactive sequence diagram. It shows the details of interaction among software components.

Initially, all operations are started by the system initialization command. After all components are initialized, TinyOS will give out a system starting command to the FairyM component, which is the central control of the application. Then the initialized timer connected to the hardware clock will periodically produce a timer event. Within a period between two timer events, processes of collecting data from hardware sensor, data encryption, synchronous signal sending, and sending data by radio, are to be done in a manner that one component issues the commands to other components and signals events to the corresponding components when some conditions are satisfied. The whole sequence is run-to-end, that is, they never stop until the power is shut down.

In order to accurately measure the running time of the TEA, we should guarantee that the message packet collected from photo and temperature sensors is broadcast to base station without much delay. However, task in TinyOS, when executed, would be interrupted by other event handlers, thus disturbing the sequence of interaction. So in our experiment, we directly implement the TEA algorithm in the message handler functions, rather than creating a task component to encrypt the message packet.

[1]This is the interactive sequence on the sensor node that encrypts message. If the encrypting operation is removed from the interactive sequence, it can be implemented on the second sensor node as a comparison.

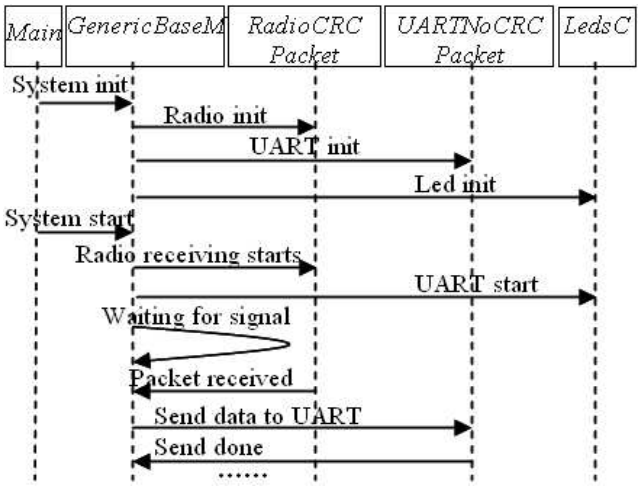

Figure 5: The sequence of interaction in the components of base station

5.4 Base Station Software Implementation

The base station captures all the packets that it can hear and transfer them to the attached PC through serial port. The base station acts as a bridge connecting resource-constrained sensor node with more powerful computer. Hence there are at lease two types of components we need to use for base station software. One is used to handle the communication with sensor node over radio. Another is to control the UART and forward packets. We implement the base station software using five components based on the platform of TinyOS:

- *Main* component. Its functionality is the same as the Main component in the sensor node.
- *GenericBaseM* component. We create this high-level component for central control of the whole system.
- *RadioCRCPacket* component. It is used to handle radio packets received from underlying hardware.
- *UARTNoCRCPacket* component. The process of forwarding packets is done by this component.
- *LedsC* component. It is the same as the LedsC component in sensor node.

Figuure 5 illustrates the interaction between components.

Similar to the implementation in sensor node, in the base station the processing begins from the initializing command issued by *Main* component. After initialization, the system becomes idle. It detects any messages from radio component to wake it up. When a message packet is detected by the radio component, it sends an event notification to the GenericBase component, which in turn, requests *UARTNoCRCPacket* component to forward the data to host PC through serial port. When forwarding is over, the *UARTNoCRCPacket* component gives back a Send Done event to high-level component: *GenericBaseM*. So the base station software performs like a transmission gateway, allowing message packets to flow from wireless sensor network to the host PC.

5.5 PC Software Implementation

The goal of program running on the host PC is to collect message packets and compute time spent by every sensor to encrypt data. As we described in the previous section, *data message* packets immediately follow the *synchronous message packets*. The *synchronous message packet* provides the starting time of the encrypting function on the sensor node, and *data message packets* indicates the ending time of the encrypting function. The program distinguishes these two types of packets and extracts appropriate time interval from them. The following are steps to process a message.

1. Receive a packet from serial communication port. The short measuring time requires the program to efficiently and accurately receive message packets and calculate the time interval.
2. Recognize the type of message packet and perform the corresponding operations. According to different type of messages, the program will record the corresponding time information into the *T1* or *T2* array. Then, the data payload of the packet is decrypted and displayed.
3. When certain amount of packets is collected, the data-collecting loop is stopped. Then we calculate time interval using formula *T2-T1* for each item in the arrays.

Due to the inherent errors in the radio transmission and restriction of motes hardware, the program should have robustness to the message loss. The program periodically receives two types of message packets. Only synchronous message packet indicates that a new time-computing loop has started. Therefore, only the data message packet with the same counter information as the immediately previous synchronous message packet can be accepted. In the implementation we use a Boolean and counter variable to guarantee this appropriate arrival order. Thus, no matter how many message packets are lost in the transmission, the process of evaluating running time always begins from the first synchronous message packet, and ends in the next data message packet with same counter ID.

5.6 Experimental Results and Evaluation

As we described in Section 5.1, a large number of message packets are periodically sent from sensor node to the host PC in a certain frequency. When a message packet is sent out, one working period is done and the next period begins. All these operations in a working period are initiated by a hardware timer event, which can interrupt other activities in the system and lead to a new period of processing. So we should guarantee that the working period should be long enough to allow microcontroller to complete all processing on message packets. If the working period is too short, the message packet is not prepared for broadcasting before it can be sent out by radio. Consequentially, the sequence of message packets will be out of order. Thus we can not obtain the accurate measurements.

Therefore, choosing the appropriate broadcasting frequency on the sensor node is critical for this experiment. Figure 6 shows how the frequency of sending message influences the measurement of time interval. X axis is the frequency of message sending and Y axis indicates the average time interval spent to receive 100 message packets. We can see that the frequency from 20 Hz to 1 Hz does not notably influence final measurement of time interval. The slight fluctuation of the curve is caused by the internal change of computing speed on the sensor node. So we can pick up any frequency in the range from 20 Hz to 1 Hz in our experiment.

For the sake of validity and efficiency, in our experiment we choose 2 Hz as our working frequency. So within 500 milli-seconds, all operations in one period will be performed.

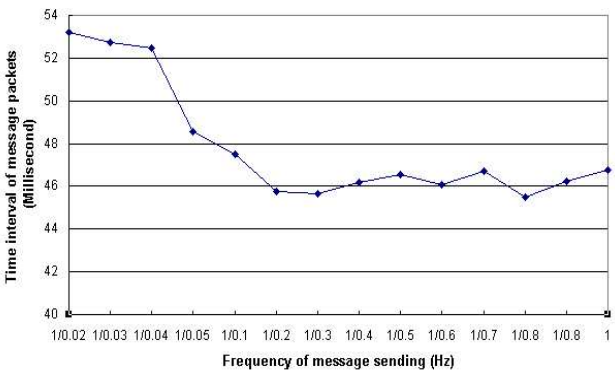

Figure 6: Average time interval in different frequencies

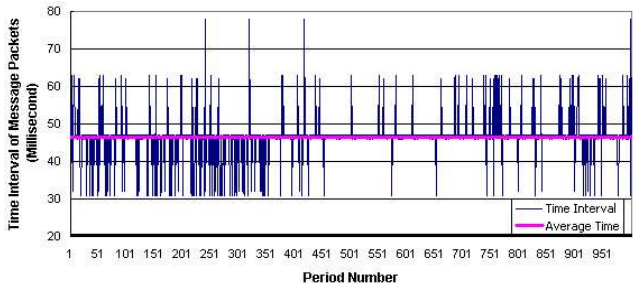

Figure 7: The measurements running the TEA

The data collected on the base station is a sequence of synchronous packets and data packets. The evaluation of running time is done by computing time interval between two types of message packets in one period.

In order to increase the accuracy of measurements, we studied a sequence that contains 1000 working periods. First, we took samples from a sensor node running the TEA algorithm. Experimental results are illustrated in the Figure 7. X axis shows the period number from 1 to 1000, and Y axis indicates measurements of time interval. Originally we expect that this line should be an ideal horizontal line without any fluctuation. However, as seen in the Figure 7 the line of measurements bounces from one period to another. Many factors could lead to this variation, such as the delay in radio transmission, the change in computing ability of the microcontroller or internal errors in TinyOS. Nevertheless all measurements are around the average time interval consistently throughout the running. Base on these data we compute the average time interval and use it as our first parameter ΔT_1 in the formula (3).

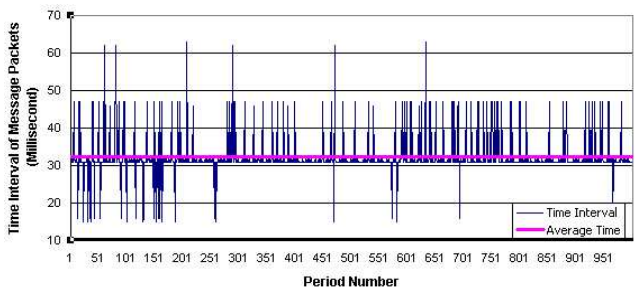

Figure 8: The measurements without the TEA

Similarly, Figure 8 shows the measurements of the sequence of 1000 periods on the comparison sensor node without running the TEA algorithm. Let us denote the average value of these measurements as ΔT_2. So the overall running time of the TEA algorithm on a sensor node is:

$$\Delta T_{TEA}=\Delta T_1-\Delta T_2=46.507-32.419=14.088 \text{ millisecond}$$

6. FUTURE WORK

Our approach of measuring running time of the TEA algorithm works well on a simple sensor network that has only one sensor and one base station. There are no complex interactive operations between sensors, such as large-scale routing, high-density communication and concurrency-intensive process inside a sensor. In order to measure the performance of encryption algorithm appropriately in a real application environment, we need to build a large sensor network with self-organized feature.

Additionally, in our experiment we use a globally shared key on both sensor node and PC machine. The key is predefined and never changed. The compromise of the sensor or PC machine will compromise the entire sensor network. Therefore, we need to implement an efficient strategy for key management on sensor network, in which the new key for participated node can be generated from the sensor master key using a pseudo-random function.

7. REFERENCES

[1] Tinyos website. http://webs.cs.berkeley.edu/tos/.

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[3] Y. Kanamori, E. Jovanov, and S.-M. Yoo. Performance comparison between tea and rijndale encryption algorithm for wireless sensor networks. In *ISCA 15th International Conference on Computer Applications in Industry and Engineering (CAINE), San Diego*, pages 209–212, Nov. 2000.

[4] J. Liu, R. K. Smith, and P. G. Bradford. Inexpensive encryption algorithms. In *Proceedings of the 41st ACM South East Regional Conference*, pages 41–44, 2003.

[5] A. Perrig, R. Szewczyk, J.D.Tygar, V. Wen, and C. David E. Spins: Security protocols for sensor networks. In *Proceedings of Mobile Networking and Computing 2001*, 2001.

[6] V. Subramonian, H.-M. Huang, S. Datar, and C. Lu. Priority scheduling in tinyos c a case study. Department of Computer Science, Washington University, St. Louis. MO.

[7] TinySec. Tinyos link layer security proposal version 1.0. http://www.cs.berkeley.edu/ nks/tinysec/design-doc.pdf.

[8] D. J. Wheeler and R. M. Needham. Tea, a tiny encryption algorithm. In *Fast Software Encryption, Second International Workshop Proceedings, SpringerVerlag*, pages 97–110, 1995.