World Academy of Science, Engineering and Technology
International Journal of Electrical and Computer Engineering
Vol:1, No:10, 2007

Implementation of TinyHash based on Hash Algorithm for Sensor Network

HangRok Lee, YongJe Choi, HoWon Kim

***Abstract*—In recent years, it has been proposed security architecture for sensor network.[2][4]. One of these, TinySec by Chris Kalof, Naveen Sastry, David Wagner had proposed Link layer security architecture, considering some problems of sensor network. (i.e : energy, bandwidth, computation capability,etc). The TinySec employs CBC_mode of encryption and CBC-MAC for authentication based on SkipJack Block Cipher. Currently, This TinySec is incorporated in the TinyOS for sensor network security.

This paper introduces TinyHash based on general hash algorithm. TinyHash is the module in order to replace parts of authentication and integrity in the TinySec. it implies that apply hash algorithm on TinySec architecture. For compatibility about TinySec, Components in TinyHash is constructed as similar structure of TinySec. And TinyHash implements the HMAC component for authentication and the Digest component for integrity of messages. Additionally, we define the some interfaces for service associated with hash algorithm.

***Keywords*—sensor network security, nesC, TinySec, TinyOS, Hash, HMAC, integrity

I. Introduction

THE sensor network is a next-generation network for moving to a ubiquitous world and many researches are achieved for realizing it. Nowadays applications using the sensor network are considered such as collection and management of environment data, emergency medical system, military service, trace and management of goods, and it is expected to expend extremely to various fields around our world.

The sensor network organizes wireless networks among sensor nodes, which are restricted in power consumption, bandwidth, memory, and calculation capability. Generally these sensor nodes communicate effectively with broadcast communication in short-range space. The wireless broadcast communication is exposed to security risks, to put it more concretely, an adversary can eavesdrop and alter communication messages, and insert malicious messages. For preventing these attacks, encryption of the communication data and mutual authentication between sensor nodes are needed. However general crypto methods, namely encryption and authentication using public-key cryptosystems, are not reasonable, because sensor nodes have very low calculation capability and small memory and they are not able to operate such crypto algorithms within sufficient time. Several researches are suggested to solve such security problems in the sensor network [2][3][4][5]. Cris, Naveen and David suggested a Link Layer Security Architecture for wireless sensor network in paper [2], which can be applicable to sensor nodes with restricted circumstances. They noticed that end-to-end secure protocols such as SSL, SSH and IPsec were inefficient for the sensor network, and implemented the Link Layer Security Architecture named by TinySec, which is based on SkipJack symmetric key crypto algorithm and operates it with CBC mode or CBC-MAC mode to provide confidentiality, integrity, and authentication. It generates secure packets by encrypting message data with group key shared among sensor nodes and calculating MAC for whole message including a header.

In this paper, we implemented security components for message hash and authentication using SHA-1 hash function instead of using CBC-MAC based on SkipJack, which is offered in TinySec. We designed security components by a similar architecture with TinySec in other to have compatibility with TinySec, we called TinyHash, and implemented interfaces for wiring these components. And then we tested execution time and memory size for SHA-1 hash function implemented over TinyHash.

This paper is organized as follows. In Section 2, the sensor network is briefly introduced and in Section 3, TinyOS and NesC are described in detail. In Section 4, the architecture and security components of TinyOS are explained. In section 5, we describe about architecture of TinyHash and show that the test results of SHA1 algorithm and HMAC implemented under TinyHash. Finally, concluding remarks and future works are presented in Section 6.

II. Sensor Networks

The sensor network implies that is organized the wireless sensor nodes with extremely limited resource. Generally, sensor network are consisted of sensor nodes for environment information sensing and base station for transaction of sensing information collected. If environment events are happened, firstly, some sensor nodes in close to environment events acquire the values of environment event, then these sensor

Manuscript received November 15, 2005

H.R Lee. is with Electronics and Telecommunications Research Institute (ETRI), 161 Gajeong-Dong Yuseong-Gu, DaeJeon, 305-350, KOREA, Tel : +82-42-860-1269, / FAX : +82-42-860-5611, e-mail : neogauss@etri.re.kr

Y.J. Choi, H.W. Kim are with Electronics and Telecommunications Research Institute (ETRI), 161 Gajeong-Dong Yuseong-Gu, DaeJeon, 305-350, KOREA, Tel : +82-42-860-1327, / FAX : +82-42-860-5611, e-mail : {choiyj,Khw}@etri.re.kr

nodes send the sensing data to base station via multi-hop route, and then base station process all sensing data received from sensor nodes. Such sensor networks can be used for various applications such as medical monitoring, habitat monitoring, inventory control, emergency response, and battlefield managements.

In sensor network, neighboring sensor nodes is high possible to aquire similar sensing information. Therefore, To reduce the wastes of sensor node's energy and bandwidth caused by broadcast mechanism in sensor network, each sensor nodes make use of in-network progressing mechanism such as aggregation and elimination of duplicate contents of received messages. Fig 1) is shown that typical structure of sensor network.

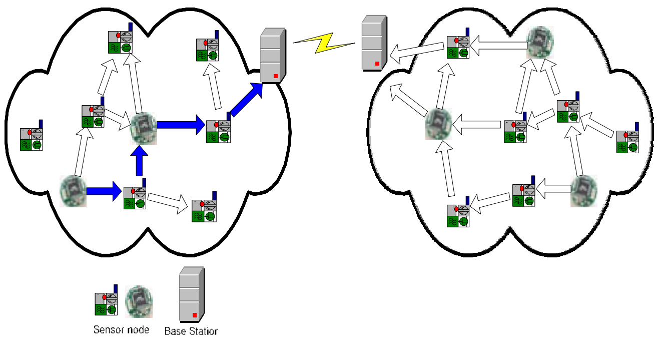


Fig. 1 Typical Structure of Sensor Network

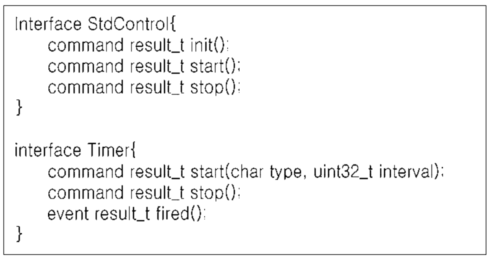
Sensor node of significant factor in sensor network is defined to device with the very small memory, Peanet CPU, and wireless Radio devide. For instance, designed by UC Berkeley, upgraded version of Mica2 mote, Telos mote is consisted of an 8MHz 16bit TI MSP430 CPU with 2KB of RAM, 60KB of EEPROM, and 2.4GHz 802.15.4 Chipcon wireless Transceiver. We will test hash algorithm and HMAC scheme with these Telos mote

III. TinyOS and NesC Language

In this section, We describe about the TinyOS and NesC Language in detail [1]. Because, we are necessary to detail understand of TinyOS and nesC language in order to explain relationship between components associated with security described in session 4 and 5. TinyOS is designed specially for small embedded systems with limited resources, and it is programming model developed to meet concept of event-driven application. The core OS requires only 400 bytes of code and memory. TinyOS had made to treat of several programming challenges in sensor network. This TinyOS has several significant features as follow.

- **Component-oriented architecture**: TinyOS provides a set of reusable system components. Application connents components using a wiring specification that is independent of component implementations.
- **Concurrency based on tasks and events**: Tasks are a deferred computation mechanism. They run to completion and do not preempt each other. If components can post tasks then it will deferring the computation until the scheduler executes the task later. Events also run to completion, but may preempt the execution of a task or another event. Events signify either completion of a split-phase operation or an event from the environment. TinyOS execution is driven by events representing hardware interrupts
- **Split-phase operations**: TinyOS has no blocking operations All long-latency operations are split-phase such that operation request and completion are separate functions. Commands are typically requests to execute an operation. For example, if the operation is split-phase, then the command returns immediately and completion will be signaled with an event.

Programming language for sensor network based on design concept of TinyOS mentioned above is nesC. NesC Langugae is based on the concept of components, and directly supports TinyOS's event-based concurrency model. NesC applications are built by wiriting and assembling components. A component provides and uses interfaces. These interfaces are the only point of access to the component. An interface generally models some service and is specified by an interface type. An interface has bidirectional properties. Namely, they contain commands and events functions declaration. In Fig 2, The Timer interface defines *start* and *stop* commands and a *fired* event.



```
Interface StdControl{
    command result_t init();
    command result_t start();
    command result_t stop();
}

interface Timer{
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event result_t fired();
}
```

Fig. 2 Some Interface Types

The Command functions in an interface call from one component requesting service from another component. Event function calls indication completion of service by a component. Since, the concept of split-phase operations in TinyOS, the any components providers implement the command functions declared in providing interface, while the component users implement the event functions declared in using interface.

As a simple example, we present Blink application that performs periodic to blink red LED on mote [9]. Blink application is organized four components together wired. Among of them, as be shown in Fig 3), BlinkM module components implements three commands in interface StdControl and one event in interface Timer.

```
module BlinkM {
    provides{
        interface StdControl;
    }
    uses{
        interface Timer;
        interface Leds;
    }
}
implementation {
    command result_t StdControl.init() {
        call Leds.init();
        return SUCCESS;
    }

    command result_t StdControl.start() {
        return call Timer.start(TIMER_REPEAT, 1000);
    }

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

    event result_t Timer.fired() {
        call Leds.redToggle();
        return SUCCESS;
    }
}
```

Fig. 3 Specification of the BlinkM module component

There is language have two types of components in nesC which is called *modules* and *configurations*. Modeles provide application code which is implemented one or mode interfaces. Configurations are used to wire other components together, connecting interfaces used by components to interfaces provided by others components. Every nesC application is described by a top-level configuration that wires together the components used.

```
configuration Blink {

}
implementation {
    components Main, BlinkM, TimerC, LedsC;
    Main.StdControl -> TimerC.StdControl;
    Main.StdControl -> BlinkM.StdControl;
    BlinkM.Timer -> TimerC.Timer[unique("Timer")];
    BlinkM.Leds -> LedsC.Leds;
}
```

Fig. 4 Specification of Blick configuration component

For example, As shown in Fig 4), Blick configuration is built by wiring the four subcomponents given by the components declaration.(Main, BlickM, TimerC, LedsC). It connects the Timer interface used by BlinkM to Timer interface provided by TimerC, and connects the Leds interface used by BlinkM to that provided by LedsC, and finally connects the StdControl interface used by both BlinkM and TimerC to that provided by Main component. In Fig5), we graphical depicts of relationship between sub-components in Blick application.

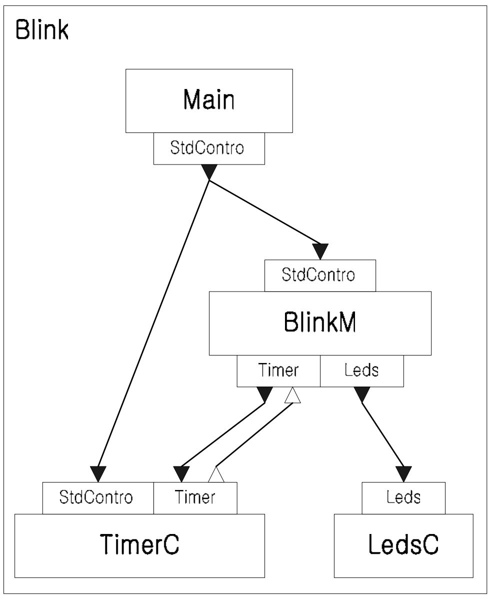


Fig. 5 graphical depiction of Blink configuration

IV. TinySec

In this section, we discuss that link-layer security architecture in sensor networks and components and interfaces associated with security in the TinySec[2]. TinySec is a link-layer security architecture based on SKIPJACK block cipher. The reason of considering link-layer security architecture for sensor network security is unlikely to use conventional the end-to-end security mechanism (i.e SSL, SSH, IPSec..) in sensor network. These facts are like to follows. The dominant traffic pattern in sensor networks is broadcast communication and many sensor nodes communicate sensor readings or network events through a multihop routing to a central base station. Also, neighboring nodes in sensor networks often reads the same or correlated environmental event values. In here, If each node sends same packets to the base station, then significant energy and bandwidth are wasted. So, To prevent these wastes, sensor networks use in-network processing such as aggregation and duplicate elimination. In order to archieve in-network processing, sensor nodes should be able to access, modify the contents of messages received. Therefore, they choose the link-layer security architecture for sensor network security, with permitting in-network processing. Link-layer security architectures ensure that authenticity, integrity, and confidentiality of messages between neighboring nodes.

TinySec architecture is shown in Fig6). TinySec supports two security options which is called TinySec-AE of authenticated encryption and TinySec-Auth of authentication only. The TinySec-AE uses the CBC mode for encryption of data payload and CBCMAC for authentication of entire packet. TinySec-Auth only use the CBCMAC for authentication of entire packet. And, TinySec is consist of five module components and only one configuration components and defines six interfaces. The key point in the TinySec is the defined interfaces for security service based on block cipher;

block cipher mode and MAC. As be described in session 3, the interfaces are the very necessary factors for modeling of providing services in TinyOS and nesC. Thus, interfaces of definitions influences usability, availability and expansion of whole structure. Therefore, we are attention to define interfaces associated with hash algorithm. Fig6) is a part of relationship between components in TinySec.

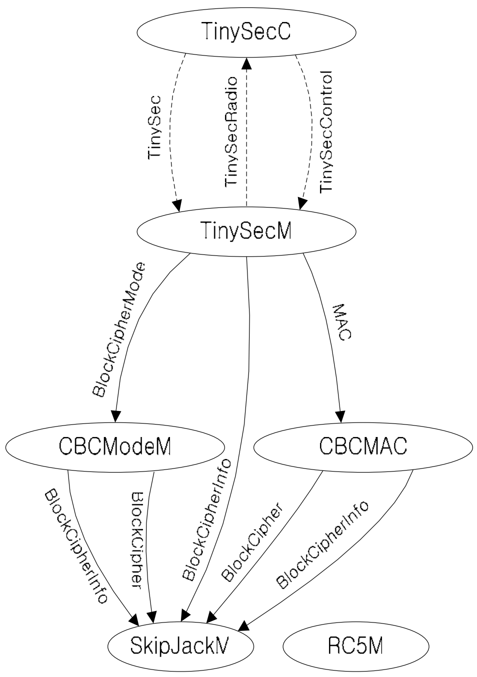


Fig. 6 Relationship between components in TinySec

V. TinyHash

In this section, we describes about the TinyHash. As mentioned above, TinySec uses CBCMAC based on block cipher for message integrity and authentication, instead of hash algorithm. Because of, The SkipJack is very light-weight block cipher algorithm and has high speed performance, then they may not implements HMAC based on hash algorithm. And also, most of hash algorithms had been developed algorithms based on 32bits operation, so it will be able to show that hash algorithm is not made display of it's original performance in CPU based on 8bits, 16bits using on typical sensor mote.

But, it exist many security of advantage due to using hash algorithms, since we will forecast to be developed light-weight hash algorithm enabling to apply in sensor node with extremely resource constraints. Therefore, we design TinyHash architecture based on hash algorithm for preparation of using in the future.

TinyHash provides integrity and authentication of message. TinyHash is applied HMAC scheme for authentication and make use of the SHA1 hash algorithm for message digest. Here, SHA1 algorithm is able to replace other hash algorithms.(i.e MD5, RIPEMD, SHA256 etc). To enable a high level application running on TinyOS to use the variety types of hash algorithm, we have implements *Digest* component to represent the hash algorithm. Because, TinyHash is independent module from the hash algorithm, not only it can easily include any hash algorithms in TinyHash architecture, but also TinyHash User can conveniently use to choose any hash algorithms. For instance, if developer will implement signature algorithm based on public key running on TinyOS, in this case, instead of directly using of specific hash algorithm in signature algorithm component, developer have only to use the *Digest* component, and only have to modify the wiring code in signature configuration component. Hence, without modifying source code in implemented application module component, applications are able to use freely some hash algorithms, only modifying of wiring in the application configuration component. The following statements are definitions of three interfaces associated with hash algorithm.

- MessageDigest : Defines commonly four command functions of hash algorithm.
  - init(), update(), updateByte(), dofinal()
- MessageDigestInfo : Defines command function of hash size.
  - getMessageDigestSize()
- HASH : Defines command functions of message digest service.
  - Init(), initIncrementalHash(), incrementalHash(), getIncrementalHash(), hash()

Since interface of HMAC service correspond to interface of MAC service in defining of TinySec, we do not defines interface of HMAC and make use of MAC interface. As be shown in Fig 7), since *HMAC* component provides MAC interface, it should implements all commands declared in MAC interface, and then *HMAC* component uses both MessageDigest and MessageDigestInfo interfaces, it should implements all events declared in that interfaces. But, defined MessageDigest and MessageDigestInfo interfaces in above, any events functions are not defined in that interfaces, therefore *HMAC* component simply should implements code to call the want to using commands in that interfaces. In Fig 8), it is shown that relationship between components in TinyHash.

We have implemented TinyHash on Telos mote, and we have implements SHA1 to 8bits version in order to have the compatibility with MICA mote series. Our Implementation of SHA1 algorithm requires 125 bytes of RAM and 4075 byte of ROM. And SHA1 performance is estimated 35ms for 160 bit message data. Also, HMAC has twice size more than SHA1 algorithm. As mentioned above, since SHA1 algorithm is designed algorithm based on 32bits, we thinks that SHA1 algorithm is not made a display of original performance on currently motes. In Table 1), we show that performance and size of TinyHash.

TABLE I
EXECUTION TIME AND SIZE ON TELOS MOTE

Implemented Algorithms	Time(ms)	Size of RAM , ROM (byte)
SHA1	35	140 , 3504
HMAC	71	165 , 4017

Table 1) Execution Time and Size on Telos mote

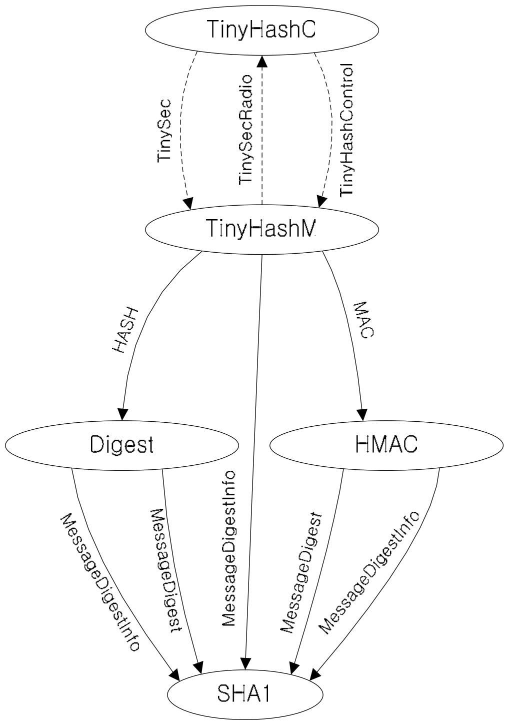


Fig. 8 Relationship between components in TinyHash

VI. CONCLUSION

In this paper, we have proposed TinyHash based on general hash algorithm providing message integrity and authentication. We have designed TinyHash to a similar architecture with TinySec in other to compatibility. And we have defined some interfaces of Message Digest services, and then we have implemented HMAC component for authentication and Digest component for integrity of message. Since TinyHash architecture is independently designed with hash algorithm, it can conveniently add and use any hash algorithms. But, as shown testing result of SHA1 in above, since previously all hash algorithms are developed algorithms based on 32 bits operation, then hash algorithms are not made a display of own natural performance in currently sensor mote. Therefore, we think that need to development of light weight hash algorithm for sensor node.

REFERENCES

[1] David Gay, Philip Levis, Robert von Behren "The nesC Language : A Holistic Approach to Networked Embedded Systems" PLDI'03 June 9-11
[2] Chris Karlof, Naveen Sastry, David Wagner "TinySec : A Link Layer Security Architecture for Wireless sensor Networks" SenSys'04 November
[3] David J. Malan, Matt Welsh, Michael D. Smith "A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography"
[4] Tieyan Li, Hongjun Wu, Xinkia Wang and Feng bao "SenSec Design" I2R Sensor Network Flagship Project : Technical Report-TR v1.0
[5] Ronald Watro, Derrick Kong,Sue-fen Cuti, Charles Gardiner, Charles Lynn, Peter Kruus " TinyPK : Securing Sensor Networks with Public Key Technology"
[6] Mihir Bellare, A. Desai, E. Jojipii, P. Rogaway. "The security of the cipher block chaining message authentication code" journal of Computer and System Sciences 61(3):362-399, December 2000
[7] FIPS PUBS 180-2 " SECURE HASH STANDARD" U.S DoC/NIST, Augest 1,2002
[8] FIPS PUBS #HMAC draft "The Keyed-Hash Message Authentication Code"
[9] "TinyOS and nesC Tutorial" http://www.tinyos.net/tinyos-1.x/doc/
[10] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. "TAG : A tiny aggregation service for ad-hoc sensor networks." In The Fifth Symposium on Operatiing Systems Design and Implementation(OSDI 2002), 2002