İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

AUTHENTICATION IN WIRELESS SENSOR NETWORKS

M.Sc. Thesis by

Mehmet Erhan YİĞİTBAŞI, B.Sc.

Department : Computer Engineering

Programme: Computer Engineering

JULY 2005

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

AUTHENTICATION IN WIRELESS SENSOR NETWORKS

M.Sc. Thesis by

Mehmet Erhan YİĞİTBAŞI, B.Sc.

(504021500)

Date of submission : 6 July 2005

Date of defence examination: 6 July 2005

Supervisor (Chairman): Prof. Dr. Bülent ÖRENCİK

Members of the Examining Committee Prof.Dr. A. Emre HARMANCI (I.T.U.)

Assoc.Prof.Dr. Erdal ÇAYIRCI (War Col.)

JULY 2005

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

KABLOSUZ DUYARGA AĞLARINDA ASILLAMA

YÜKSEK LİSANS TEZİ
Müh. Mehmet Erhan YİĞİTBAŞI
(504021500)

Tezin Enstitüye Verildiği Tarih : 6 Temmuz 2005
Tezin Savunulduğu Tarih : 6 Temmuz 2005

Tez Danışmanı : **Prof. Dr. Bülent ÖRENCİK**
Diğer Jüri Üyeleri **Prof.Dr. A. Emre HARMANCI (İ.T.Ü.)**
Doç.Dr. Erdal ÇAYIRCI (Harp Akademileri)

TEMMUZ 2005

FOREWORD

First, I would like to thank my advisor Prof. Bülent Örencik for his support and guidance during my master's program. He introduced new areas of research and helped me to understand the approach to solve research problems. Next, I would like to thank my guidance committee members for their valuable comments on my thesis.

Finally, I would like to thank my parents for their love and affection. I thank them for their support in whatever efforts I make.

Mehmet Erhan YİĞİTBAŞI

July 2005

TABLE OF CONTENTS

ABBREVATIONS

DARPA	: American Defense Advanced Research Projects Agency
PDA	: Personal Digital Assistant
WLAN	: Wireless Local Area Networks
IEEE	: Institute of Electrical & Electronic Engineers
MAC	: Medium Access Control, Message Authentication Code
RTS	: Request to Send
CTS	: Clear to Send
IV	: Initialization Vector
EEPROM	: Electrically Erasable Programmable Read Only Memory
ROM	: Read Only Memory
RAM	: Random Access Memory
BS	: Base Station
QoS	: Quality of Service
CA	: Certificate Authority
DSS	: Digital Signature Standard
TTP	: Trusted Third Party
SAP	: Space Authentication Portal
AS	: Authentication Server
TGS	: Ticket Granting Server
SEAL	: Self Authenticating Value
LHAP	: A Lightweight Hop by Hop Authentication Protocol
GPS	: Global Positioning Systems
LEAP	: Localized Encryption and Authentication Protocol
CBRP	: Cluster Based Routing Protocol
DTG	: Date Time Group
ARM	: Advanced RISC Machines
PRG	: Pseudo Random Number Generator
CVS	: Concurrent Version System
DES	: Data Encryption Standard
NSA	: National Security Agency
PC	: Personal Computer
UART	: Universal Asynchronous Receiver Transceiver
SNEP	: Security Network Encryption Protocols
TESLA	: Timed Efficient Stream Loss Tolerant Authentication
µTESLA	: Micro Timed Efficient Stream Loss Tolerant Authentication

TABLE LIST

FIGURE LIST

Page Number

SYMBOL LIST

F	**:** Pseudo random functions
H, h	**:** Hash functions
G	**:** Hash function family
K_i, K(i)	**:** i^{th} key
K^F_A	**:** Traffic key
K^T_A	**:** Tesla key
T, t	**:** Time

KABLOSUZ DUYARGA AĞLARINDA ASILLAMA

ÖZET

Düşük güç tüketimli duyargalardan oluşturulmuş kablosuz ağlar yakın gelecekte pek çok yerde kullanılır hale gelecektir. Hatta geliştirilecek olan uygulamaların birçoğu kablosuz duyarga ağlarını hedefleyecektir. Gerçekleştirilecek bu çalışmalardan askeri alandan sağlık ve çevre alanına kadar pek çok sahada faydalanılacaktır. Nano teknoloji, mikro elektronik ve mekanik sistemler, radyo haberleşmesi ve mikroişlemci dünyasındaki birçok gelişme de duyarga ağlarının yaygınlaşması sağlamış ve kullanılabilirliğini arttırmıştır.

Kablosuz duyarga ağlarındaki duyargaların sayılarının çokluğu ve bu cihazların kısıtlı kaynaklar ile olabildiğince düşük güç tüketecek şekilde yönetilmesi gerekliliği bu ağlarda güvenli iletişim sağlamayı zorlaştırmaktadır. Genel güvenlik servislerinin (gizlilik, asıllama, bütünlük vs.) sağlanabilmesi için, yapılan çalışmalarda bu duyargaların kaynak kısıtları büyük ölçüde göz önünde bulundurulmalıdır.

Yapılan tez çalışmasında önerilmiş veya gerçekleştirilmiş kablosuz duyarga ağları asıllama protokolleri incelenmiş ve ayrıntılarıyla açıklanmıştır. Son olarak, tamamıyla gerçeklenmiş olan ilk kablosuz duyarga ağları veri bağı katmanı asıllama protokolü TinySec incelenmiştir. TinySec duyargalardaki kaynak kısıtlarına bağlı olarak TinyOS işletim sistemi üzerinde geliştirilmiş, birçok donanımla uyumlu çalışabilen bir protokoldür.

TinySec alınan mesajların asıllanabilmesi için mesajların sonuna kapalı anahtarla hesaplanan ve bir şifreleme algoritmasına dayanan mesaj asıllama kodu eklemektedir. Tez çalışmasında mesaj asıllama kodu hesaplanmasında kullanılan şifreleme algoritması ve altyapı değiştirilerek, performans karşılaştırılması yapılmıştır. RC5 ve Skipjack algoritmaları kullanılarak yapılan karşılaştırmalarda RC5 ile yapılan asıllamanın daha hızlı olduğu ve daha az güç tükettiği sonucuna varılmıştır.

AUTHENTICATION IN WIRELESS SENSOR NETWORKS

SUMMARY

Wireless networks of low-power sensing devices are going to be used in everywhere in the near future. These wireless sensor network applications will become a ubiquitous part of the computing landscape. Proposed applications of these networks range from health care to warfare. The emergence of low-power sensor networking has been propelled by the convergence of advances in several fields, including nano technology, Micro Electronic Mechanical Systems (MEMS), radio frequency communications, and microprocessors.

The potentially unlimited size of a sensor network with thousands of nodes and the need to manage with limited resources and conserve energy as much as possible, on each single node as well as throughout the network, makes secure communication challenging. Nodes depend on each other for correct operation. Messages have to be transmitted over several hops, since direct communication between arbitrary nodes is impossible due to limited radio range. Nodes have little knowledge of other, distant nodes. The challenge for the information security community is to develop the common security services (confidentiality, integrity, etc.) for sensor networks in a manner that meets the very strict resource constraints of these devices.

In this thesis, a broad range of on-going research efforts in authentication within the wireless sensor networks are described in detail. Finally, TinySec which is said to be the first fully-implemented link layer security architecture for wireless sensor networks is discussed which addresses sensors' resource constraints with a good design and which is also portable to a variety of hardware and radio platforms as it uses TinyOS.

TinySec uses message authentication codes for authentication which are formed by secure hash functions and an encryption algorithm. In this thesis TinySec's underlying authentication and encryption mechanism is changed and compared by using two different encryption algorithms which are RC5 and Skipjack. It is seen that using RC5 for authentication within TinySec is slightly faster than using Skipjack and it consumes less power.

1. INTRODUCTION

Today microprocessors are embedded in nearly all electronic devices such as cellular phones, televisions and camcorders. In combination with computing devices such as desktop computers, notebooks and personal digital assistants (PDAs) this could form a huge wireless network of mobile and static devices communicating without fixed infrastructure or centralized administration. In such a self-organized network each node relies on its neighbor nodes to keep the network connected. The security issues for ad-hoc networks are different than the ones for fixed networks. System constraints include low power microprocessor, small memory, small bandwidth, and limited battery power.

"*An ad-hoc network is a wireless network made up of mobile hosts that do not require any fixed infrastructure to communicate. As early in the 1970s, ad-hoc networks were called packet radio networks which were investigated for military applications and were developed by the DARPA*" [1]. When designing the 802.11 standard for WLAN, the IEEE replaced the term packet radio network by ad-hoc network. Ad hoc networks are frequently associated with self organization, which means that they run solely by the operation of end users.

Communication links are wireless to guarantee mobility. Ad-hoc networks act independently from any provider. The network topology may be very dynamic, making the links and routes very unstable. Power management is an important system design criteria. Finally, security is a critical issue because of the weak connectivity and of the limited physical protection of the mobile hosts.

Because packet radio networks have been developed initially for military purposes, potential applications are very often associated with critical situations such as battlefields and damaged areas. Other prospective applications are still at an early stage, as military applications like communications between soldiers, soldiers monitoring, sensor networks for target detection and identification.

Other examples can be emergency situations, where the existing network infrastructure is not reliable and distributed networks for data collection and device monitoring like sensor networks, home networks and inter-vehicle communications. There can be need for creation of temporary networks for ad-hoc meetings, conferences or brainstorming. Most of the application areas are health, military, and security. For example, the physiological data about a patient can be monitored remotely by a doctor [2].

Realization of these and other sensor network applications require wireless ad hoc networking techniques. However sensor networks differ from ad hoc networks with some different characteristics. The number of sensor nodes in a sensor network can be several orders of magnitude higher than the nodes in an ad hoc network where sensor nodes are densely deployed. The topology of a sensor network changes very frequently and sensor nodes mainly use broadcast communication whereas most ad hoc networks use point-to-point communications. Sensor nodes are limited in power, computational capacities and memory [2].

The sensors in the sensor network make measurements, such as local temperature or pressure, and share this data with the appropriate application via the network. Providing security mechanisms for sensor networks is a critical issue since sensors will be used everywhere. Authentication of the data source and the data are critical concerns since somebody might attempt to capture sensors and tamper with sensor data.

Traditional authentication frameworks based on public key cryptography are not suitable for sensor networks since the sensors consist of small, low-powered microcontrollers that are mobile. The resources of the sensor are limited and as a result of this, the authentication and encryption mechanisms for sensor networks should be very lightweight.

2. SENSOR NETWORK ARCHITECTURE

Various factors should be considered in design of a sensor network. Environment, data types, communication layers and communication architecture are the important factors for a sensor network architecture.

2.1 Environment

The sensor network environment can be physically demanding on sensor nodes. The communications environment may be problematic due to interference and fading due to ground placement. "*The population density of sensor nodes in the network may vary depending upon the application, the communication capabilities of the sensor nodes, and the environment (e.g., desert, rain forest)*" [3]. For example, a one-dimensional boundary application may require sensor nodes be placed every 100 meters in a line. The same application may require a closer placement in a denser terrain that limits signal propagation. In most scenarios, it is assumed that once deployed sensor nodes have no mobility which implies that the network is somewhat static. However, although nodes are not mobile, the topology of the network may change as nodes are added or deleted from the network. Nodes may be added to replace nodes that have lost power or were destroyed.

2.2 Data Types

Within the sensor network, the amount and type of data exchanged is greatly influenced by the battery and energy constraints of the sensor nodes. The energy required to transmit a bit can be much greater than the cost to internally process a bit. For this reason, raw data will typically be processed locally and the results are exchanged within the network with fewer transmitted bits and as a result of this less energy will be consumed. The data exchanged within the sensor network may

include raw sensor data, sensor node event reports destined for a remote command center or dismounted soldier in the field, or sensor commands and controls [4].

In order to conserve energy, raw sensor data is not usually forwarded within the network but processed locally into event reports that may include target classification and direction information. Data fusion helps to reduce the total amount of bits of data routed within the network. For some sensor applications it is assumed that raw real time data such as voice or video may be exchanged without significant local processing.

2.3 Communications Architecture and Layers

The distributed sensor network is an ad hoc wireless network where the membership and roles of sensor nodes is generally not known until the deployment of the network. Sensor nodes may be deleted permanently from the network when their available energy falls below acceptable limits or temporarily when they return to a sleep state. Once deployed, the network is self-organizing, developing a routing topology that provides strong connectivity throughout the network.

In order to maintain the energy balance within the network, re-organization is required throughout the life of the network as nodes are deleted and added to the network. This creates a fault tolerant network design where the loss of a fraction of the nodes causes a graceful degradation in network performance.

It is assumed that the sensor network supports a layered protocol stack as shown in Figure 2.1 [4]. The physical layer provides a wireless link between neighboring sensor nodes while the network layer allows for routing and delivery of data throughout the network. Some applications may require reliable delivery services from a transport layer and various sensor applications are supported at the application layer.

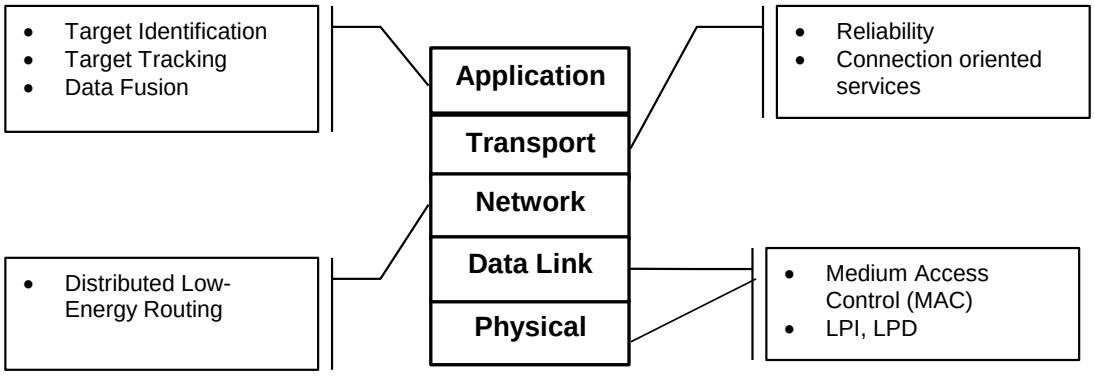


Figure 2.1: Sensor Networks Communication Layers [4]

2.3.1 Physical Layer

The physical layer defines the mechanisms for medium access control (MAC) for the wireless sensor network. There are various physical layer MAC protocols that may be used for sensor routing. It is assumed that nodes have variable control over their radiant RF energy allowing them to dynamically control the range of their communications and provide a lower probability of interception [5].

2.3.2 Network Routing Layer

The sensor network utilizes a multihop packet based network routing protocol to deliver data throughout the network. "*The finite energy of the network is the primary design constraint in developing a low-energy routing algorithm that balances energy throughout the network. Nodes that are a focal point for network traffic will lose energy more quickly than those nodes at the edges of the network*" [5].

Self-organization is required at time of deployment to initialize routing tables without the assistance of a human administrator. Re-organization is also necessary because of the ad hoc nature of the network and sensor nodes may be added and deleted from the network over its lifetime.

As a multi-hop network, packets are transferred from node to node until they reach their final destination. Intermediary nodes make routing decisions based on their routing tables that are constructed based on link costs that consider the energy to transmit and receive [3,5].

2.3.3 Transport Layer

The transport layer protocols can provide reliability and session control for sensor node applications. The majority of sensor network communications are packet oriented communications that do not require the reliability of the transport layer. Using real-time multimedia applications over the sensor network may require the ordering and reliability mechanisms of transport layer protocols.

3. SENSOR NODE TECHNOLOGY

The sensor node is the basic component of the sensor network. Nodes are designed for ease of deployment and to be low cost, compact, lightweight, and disposable. Local and collaborative signal processing across the wireless network enhances sensor nodes primitive sensing functions (e.g., seismic, acoustic, magnetic).

3.1 Sensor Node Hardware

Sensor nodes provide the core sensing functions of the sensor network. The sensor node hardware design and communications architecture are greatly influenced by their finite battery limitations.

3.1.1 Hardware Design

The exact function of each sensor node may not be determined until deployment and may change over the course of its mission. "*Flexibility is an important requirement in reducing the amount of equipment needed in order to deploy a sensor network and in supporting remote deployment techniques. They can be remotely re-programmable to support new functionality and they can support location determination mechanisms to define their exact or relative position*" [4].

Energy is the most constraining factor, affecting all aspects of a sensor node design. Microprocessor selection is one area where energy conservation is important. There are numerous commercially available microprocessors designed for embedded low power environments. These microprocessors are suitable for both commercial applications and sensor nodes with similar energy constraints [4].

3.1.2 Sensor Node Energy

The greatest limiting factor in a sensor node's life expectancy is its battery capacity. Energy conservation is applicable at the node level and at the network level. Once

deployed, the sensor node battery cannot be recharged or replaced while in the field. However, some developers are investigating using solar arrays to add recharging capabilities. "*The actual amount of energy available by a sensor node's battery is a function of temperature, the rate of dissipation, and the battery technology. A battery's potential capacity is measured in milliampere-hours (mAh) and is a function of the temperature and the load on the battery*" [4].

In order to conserve energy, embedded processors typically have low power modes that slow or halt the processor clock and place the device in a state that consumes less power. In most of the developed sensor nodes the microprocessor is switched to the sleep mode when it has no data to process.

3.1.3 Sensor Node Mobility

In most of the designed systems it is assumed that once deployed, the sensor nodes will not be physically moved. However, in some environments, sensor nodes may be replaced if the battery supplies are low or the nodes are damaged. The topology of the network changes over time as nodes are added or deleted from the network.

3.1.4 Sensing Capabilities

Sensors may contain any number of sensing capabilities including seismic, acoustic, magnetic, infrared, radar, and video. "*Although the sensing of events is done in real time, the reporting of events may not be in real time. Reports may be fused with reports from other sensor nodes. Sensor nodes may deliver video in real time and require suitable QoS to support its delivery"* [4].

3.2 Software

Mostly sensor nodes employ an embedded operating system to manage and support its applications for providing real-time performance. The operating system should be trustworthy. In order to support the implementation of any security requirements, the embedded operating system is not bypassable. Sensor nodes can support remote reconfiguration and reprogramming to incorporate flexibility.

4. SECURITY REQUIREMENTS

Security requirements for a sensor network are similar to other network types. Generally confidentiality, authenticity, integrity, non-repudiation, freshness and scalability should be considered. Also availability, accessibility, self organization and flexibility can be an important requirement for sensor networks.

4.1 Confidentiality

The data must be protected from disclosure to authorized parties. Similarly, public sensor information, such as sensor identities and public keys, should also be encrypted to protect against traffic analysis. Confidentiality should be provided by keys with as small a scope as possible to discourage a single break from compromising a large portion of the sensor network. Establishing unique keys between every pair of communicating sensor nodes is more secure than using a single network wide key.

4.2 Authenticity

At a minimum the access to the sensor data should be limited to only those parties identified in the protocol, (e.g. implicit key authentication or data origin authentication of the shared key).

4.3 Integrity

The sensor data must not be modified by the actions of outsiders of the protocol. In other words, an adversary should not be able influence the value of the keys or the data.

4.4 Non Repudiation

"*Non repudiation is an attribute of communications that seeks to prevent future false denial of involvement by either party. Non repudiation with proof of origin provides the recipient of sensor data with evidence that proves the origin of the data. Non repudiation with proof of receipt provides the originator of the sensor data with evidence that proves the data was received as addressed*" [5].

4.5 Freshness

A protocol ideally should guarantee its participants that each shared key or the data is fresh and not been reused or resent.

4.6 Scalability

Large networks can not utilize a protocol that has poor scaling properties. The system should be scalable to be used in large networks.

4.7 Availability

Services must ensure that confidentiality and group level authentication services are available to authorized parties when needed, protecting against active attacks that attempt to interrupt service within the network. To ensure the availability of message protection, the sensor network should protect its resources (e.g. sensor nodes) from unnecessary processing of messages in order to minimize energy consumption and extend the life of the network.

4.8 Accessibility

End to end confidentiality of sensor data should not be performed since it prevents sensor data fusion by intermediate nodes from taking place. An effective technique to extend sensor network lifetime is to limit the amount of data sent back to reporting nodes. Limiting this data reduces communications energy consumption [4].

4.9 Self Organization

As distributed sensor networks must self organize their routing, they must also self organize their key management. "*The self organizing capability of sensor networks must also be able to deal with nodes failing during deployment or at other times during the lifetime of the network*" [4]. These failures may be caused by energy exhaustion, adversary actions like jamming and capture or through natural causes.

4.10 Flexibility

Some sensor networks will be used in dynamic battlefield scenarios where environmental conditions, threat and mission may change rapidly. Changing mission goals may require sensors to be removed from or added to an established sensor node. Furthermore, two or more sensor networks may be fused into one or a single network may be split in two [4].

5. DENIAL OF SERVICE THREATS IN NETWORK LAYERS

A DoS attack is any event that diminishes or eliminates a network's capacity to perform its expected function. Hardware failures, software bugs, resource exhaustion, environmental conditions, or any complicated interaction between these factors can cause DoS (see Table 5.1).

Table 5.1: Sensor network layers, denial of service attacks and defenses.

Network layer	Attacks	Defenses
Physical	Jamming, Tampering	Spread spectrum, priority messages, lower duty cycle, region mapping, mode change, Tamper proofing, hiding
Link	Collision, Exhaustion, Unfairness	Error-correcting code, Rate limitation, Small frames
Network and routing	Neglect and greed, Homing, Misdirection, Black holes	Redundancy, probing Encryption Egress filtering, authorization, monitoring Authorization, monitoring, redundancy
Transport	Flooding, Desynchronization	Client puzzles, Authentication

5.1 Physical Layer

Nodes in a sensor network use wireless communication because the network type is ad hoc, large-scale deployment makes anything else impractical. Base stations or uplink nodes can use wired or satellite communication, but limitations on their mobility and energy make them scarcer.

A well-known attack on wireless communication, jamming interferes with the radio frequencies of the nodes. For single-frequency networks, this attack is simple and effective. A node can easily distinguish jamming from the failure of its neighbors by determining the constant energy [6].

An attacker can damage or replace sensor and computation hardware or extract sensitive material such as cryptographic keys to gain unrestricted access to higher levels of communication. One defense involves tamper proofing the node's physical package. When possible, the node should react to tampering in a fail complete manner, for example it could erase cryptographic or program memory. Other traditional physical defenses include camouflaging or hiding nodes [6].

5.2 Link Layer

The link or media access control (MAC) layer provides channel arbitration for neighbor to neighbor communication. Cooperative schemes that rely on carrier sense, which let nodes detect if other nodes are transmitting, are particularly vulnerable to DoS.

Adversaries may only need to induce a collision in one octet of a transmission to disrupt an entire packet. A change in the data portion causes a checksum mismatch, a corrupted ACK control message causes back-off. Error correcting codes provide a mechanism for tolerating variable levels of corruption in messages [6].

Constant transmission of some packets would eventually exhaust the energy resources of both nodes. For example, IEEE 802.11-based MAC protocols use request to send (RTS), clear to send (CTS) to reserve channel access. To exhaust the energy resources the node could repeatedly request channel access with RTS, and get a CTS response from the targeted neighbor. [2].

5.3 Network and Routing Layer

Higher layers may not require fully reliable transmission streams, but the network layer provides a critical service. In a large-scale deployment, messages may traverse many hops before reaching their destination. As the aggregate network cost of

relaying a packet increases, the probability that the network will drop or misdirect the packet along the way also increases.

One simple form of DoS attacks the node as router vulnerability by arbitrarily neglecting to route some messages. The malicious node can still participate in lower level protocols, and may even acknowledge reception of data to the sender, but it drops messages on a random or arbitrary basis where such a node is neglectful. If it also gives undue priority to its own messages, it is also greedy [6].

In most sensor networks, some nodes will have special responsibilities, such as being elected the leader of a local group for coordination. These nodes attract an adversary's interest because they provide critical services to the network.

A more active attack, misdirection, forwards messages along wrong paths. As a mechanism for diverting traffic away from its intended destination, this DoS attack targets the sender. By verifying the source addresses, parent routers can verify that all routed packets from below could have been originated legitimately by their children.

Nodes advertise zero cost routes to every other node, forming routing black holes within the network [6]. As their advertisement propagates, the network routes more traffic in their direction. In addition to disrupting message delivery, this causes intense resource contention around the malicious node as neighbors compete for limited bandwidth. These neighbors may themselves be exhausted prematurely, causing a hole or partition in the network. One defense against misdirection and black-hole attacks let only authorized nodes exchange routing information.

Nodes can also monitor their neighbors to ensure that they observe proper routing behavior. In one approach, the node relays a message to the next hop and then acts as a watchdog that verifies the next-hop transmission of the same packet [6].

Networks using geography based routing can use knowledge of the physical topology to detect black holes by periodically sending probes that cross the network's diameter. Subject to transient routing errors and overload, a probing node can identify blackout regions.

"*Redundancy can lessen the probability of encountering a malicious node. The network can send duplicate messages along the same path to protect against intermittent routing failure or random malice. If each message uses a different path, one of them might bypass consistently neglectful adversaries or even black holes. A more clever approach uses diversity coding to send encoded messages along different paths, but with lower cost than full duplication*" [6].

5.4 Transport Layer

This layer manages end to end connections. The service the layer provides can be both simple or complex and costly. Sensor networks tend to use simple protocols to minimize the communication overhead of acknowledgments and retransmissions.

Protocols that must maintain state at either end are vulnerable to memory exhaustion through flooding. Limiting the number of connections prevents complete resource exhaustion, which would interfere with all other processes at the victim. However, this solution also prevents legitimate clients from connecting to the victim, as queues and tables fill with abandoned connections. Protocols that are connectionless, and therefore stateless, can naturally resist this type of attack.

An existing connection between two end points can be disrupted by desynchronization. In this attack, the adversary repeatedly forges messages to one or both end points. These messages carry sequence numbers or control flags that cause the end points to request retransmission of missed frames. If the adversary can maintain proper timing, it can prevent the end points from exchanging any useful information, causing them to waste energy in an endless synchronization recovery protocol.

One counter to this attack authenticates all packets exchanged, including all control fields in the transport protocol header. Assuming that the adversary also cannot forge the authentication mechanism, the end points could then detect and ignore the malicious packets.

6. SENSOR NODE CONSTRAINTS

6.1 Battery Power and Energy

Energy is perhaps the greatest constraint to sensor node capabilities and once sensor nodes are deployed in a sensor network, they cannot be recharged. Therefore, the battery charge taken with them to the field must be conserved to extend the life of the individual sensor node and the entire sensor network [2]. Various mechanisms within the network architecture, including the sensor node hardware, take this limitation into account. When considering implementing a cryptographic function or protocol within a sensor node, the impact on the sensor node's available energy must be considered.

The extra power consumed by sensor nodes due to security is related to the processing required for security functions, the energy required to transmit the security related data and the energy required to store security parameters in a secure manner. Since the amount of additional energy consumed for protecting each message is relatively small, the greatest consumer of energy in the security realm is data transmission.

6.1.1 Computational Energy Consumption

The amount of computational energy consumed by a security function on a given microprocessor is primarily determined by the processor power consumption, the processor clock frequency, and the number of clocks needed by the processor to compute the security function. The cryptographic algorithm and the efficiency of the software implementation determine the number of clocks necessary to perform the security function.

Public key cryptographic algorithms such as RSA are computationally intensive, executing thousands or even millions of multiplication instructions to perform a

single security operation. Thus, a microprocessor's public key algorithm efficiency is primarily determined by the number of clocks required to perform a multiply instruction. Table 6.1 shows the wide variance of energy consumption for representative embedded microprocessors in computing a basic public key algorithm building block, a multiply function with a 128-bit result [4].

Table 6.1: Computation Time and Energy Consumption for 128-bit Multiply Result [4]

Processor	Power Consumption (mW)	Clock Freq. (MHz)	Native Mult. Result	# clocks to compute128 bit result	Time required (μs)	Energy consumed (nJ)
MIPS R4000	230	80	128	40	0.50	115.0
SA-1110 "Strong ARM"	240	133	64	60	0.45	108.0
Z-180	300	10	32	912	91	27000.0
MC68328 "Dragon Ball"	52	16	32	1920	120.	6200.0
MCF5204 "Cold Fire"	625	33	32	304	9.2	5800.0
MMC2001"M-Core"	81	33	32	416	12.6	1020.0
ARC 3	2	40	32	168	4.2	8.4

6.1.2 Communications Energy Consumption

In addition to consuming energy through computational processing, security functions also consume energy due to the communication of information between sensor nodes. Communications energy consumption attributable to security includes exchange of key management information, including encrypted keys, certificates, and nonces, per-message additions, including initialization vectors (IVs), encryption padding, authentication tags, and signatures.

6.2 Sleep Patterns

In order to conserve energy sensor nodes spend most of their time in low power sleep modes and only awake when required to processes an event. For this reason, a node's availability within the sensor network may be limited. The result of these sleep patterns is potential unavailability of a node to receive data. In order to maintain cryptographic synchronization throughout the sensor network, it is essential that all nodes use the proper cryptographic material when communicating.

6.3 Transmission Range

The communications range of sensor nodes is limited in order to conserve energy. Reducing the transmission power saves sensor node energy and provides a lower probability of detection. The actual range achieved from a given transmission signal strength is dependent on various environmental factors.

6.4 Memory

Sensor processors require different types of memory to perform various processing functions. ROM or EPROM is needed for storing the general purpose programming such as an embedded operation system, security functions, and basic networking capability. RAM is needed for storing application programs, sensor data, and intermediate computations. Programmable memory such as EEPROM and FLASH are needed for storing downloaded application code, data between sleep periods [4].

6.4.1 Program Storage and Working Memory

The amount of program storage available for storing security functionality, such as security mechanism implementations, is unlikely to be a constraining factor on security design. Even the most sophisticated cryptographic algorithms can be represented in the tens of kilobytes of memory, whereas the amount of program storage available in ROM, EEPROM, or other nonvolatile memory is likely to be in the hundreds of kilobytes or megabytes.

It is stated that the amount of working memory available for security functionality can also not be a constraining factor. Most symmetric encryption and hashing functions can be executed in less than one kilobyte of RAM. Even the more memory consuming public key functions can be executed in just a few kilobytes of RAM [4].

6.4.2 Programmable Storage for Security Information

Key management functions often require some form of programmable memory to store long term symmetric, public, or private keys. Depending on the concept of operations, security architecture, and memory technology, programming may take place during manufacture, during pre-deployment, or even when deployed on the battlefield.

6.5 Location Sensing

The sensor network environment may not be supportive of satellite location determination technologies like GPS. "*GPS may not be well suited for environments that shield its signals (e.g., inside a building). Other technologies like the Localizer enable sensor nodes to determine relative position to other sensor nodes*" [7].

6.6 Time

Time data within the sensor network is required for synchronization of events. Time synchronization messages issued from a time source must be resistant to modification attacks in order to maintain network synchronization of events. For example, sensor node event reports can be time critical. An alteration of their time stamps may change the significant of the report. GPS offers a non-spoofable time source. If GPS is not available, other methods can be used to maintain time within the network. This includes accurate local clocks or network time protocols that synchronize time across a network.

7. NETWORKING CONSTRAINTS

Sensor networks are ad hoc in nature with the composition of the network determined at the time of deployment. During the sensor node mission, the composition of the network and its routing topology may change. This constraint limits ability to pre-configure sensor nodes for specific purposes. Sensor nodes should be able to support various roles in the network to ensure the reliability of the network.

7.1 Data Rate and Packet Size

Both the data rate and packet size affect the overall sensor node energy consumption. Generally packet sizes within the sensor network are relatively small, potentially as small as 30 bytes with header and the data rates are relatively low, less than 1 kbit/second.

7.2 Channel Error Rate

Low-layer communications protocols offer error detection and correction services. Errors that propagate into the layers where confidentiality, integrity, or authentication services are applied will affect their verification and authentication processes preventing any application data from being exchanged.

7.3 Unreliable Communications

The packet-based routing of the sensor network is connectionless and unreliable. Packets may get damaged due to channel errors or dropped at highly congested nodes. Connection oriented transport protocols such as TCP may be added for reliability. Reliability is required for the distribution of key material and security critical commands [4].

7.4 Latency

The multihop routing of the sensor network introduces delay within the network. Congestion and node processing can be a factor to the amount of latency in the network. For critical event reports and cryptographic key distribution, latency should be kept to a minimum in order to insure the timeliness of the data.

7.5 Frequent Routing Changes

As the available energy decreases in key nodes throughout the network, the need to change the routing topology to balance the energy usage within the network becomes important. Frequent routing changes can mean that the intermediate nodes processing data for end to end session can change.

8. CLASSIFICATION OF AUTHENTICATION MECHANISMS

Authentication exists to establish trust between two parties, or authentication entities. These entities consist of an identity and a key. Authentication is established by performing a cryptographic operation on the parties' identities and keys. The cryptographic operation, or authentication algorithm, establishes the nature of the trust between the parties.

Authentication algorithms can be classified according to the model they use, according to the communication organization and network topology, according to the key management system they use, according to the replay attacks protection or according to the involved parties in the algorithm.

8.1 According to Authentication Models

Some of the algorithms use direct authentication methods and some of them use threshold cryptography (Figure 8.1).

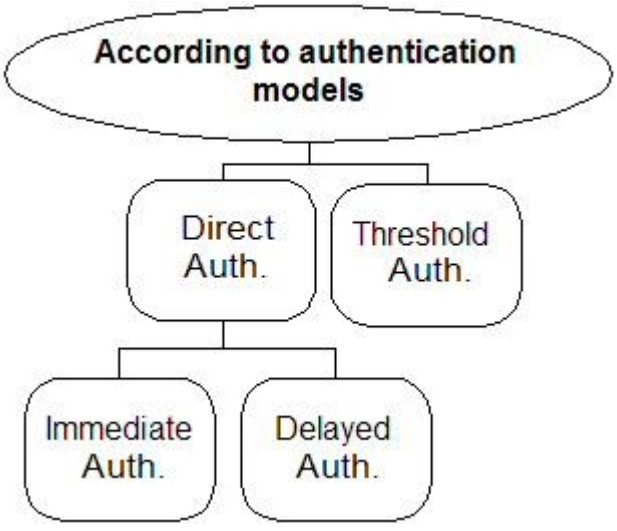


Figure 8.1: Classification of authentication schemes according to authentication models

8.1.1 Direct Authentication

In this model authentication is done within the two parties that are communicating directly and this process can be done immediately or delayed.

8.1.1.1 Immediate Authentication

In Some kind of protocols and frameworks authentication is done immediately when packet is received. Generally the required data for authentication is appended in the current packet.

8.1.1.2 Delayed Authentication

Some of the protocols does not enable immediate authentication of the packets. Authentication can only be done after some amount of time. For example after the disclosure of the key in the key chain.

8.1.2 Threshold Authentication

Authentication is done according to the other nodes in the network. For an (n,t) threshold authentication t of the n nodes must trust and authenticate the node for authentication to be accepted.

8.2 According to Communication Organization and Network Topology

The network topology and the architecture of the network also affect the authentication protocols. The nodes in the network can be static or mobile. And also the network can be supposed to operate cluster based or hierarchical. For example some of the authentication protocols are only designed for static and hierarchical networks (Figure 8.2).

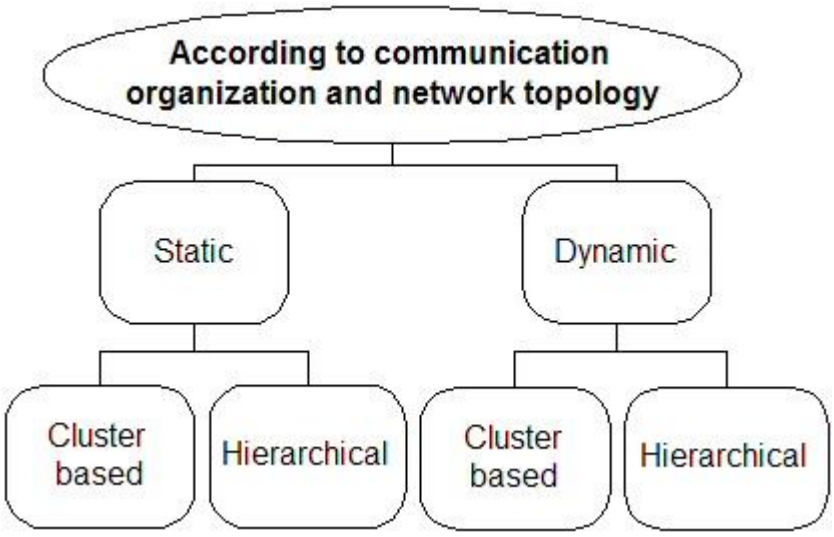


Figure 8.2: Classification of authentication schemes according to communication organization and network topology

8.2.1 Static

It is supposed that the nodes in the network never move and the topology generally does not change. The network can also be hierarchical or cluster based. In cluster based protocols communication in the network is based on clusters and some special routing protocols can be used. Some of the protocols assume a hierarchical network where all the data flow is from the sensor nodes to the application or a base station.

8.2.2 Dynamic

It is supposed that the nodes in the network can move and the topology can change. Within this type of topology hierarchical and cluster based communication can also be considered.

8.3 According to Key Management Systems

Authentication mechanisms and algorithms can be classified according to the key management systems that they use. First, there is symmetric authentication where symmetric key management schemes are used and secondly, there is asymmetric authentication where asymmetric key management schemes are used. Finally, there exists a hybrid authentication where both of the symmetric and asymmetric schemes can be used (Figure 8.3).

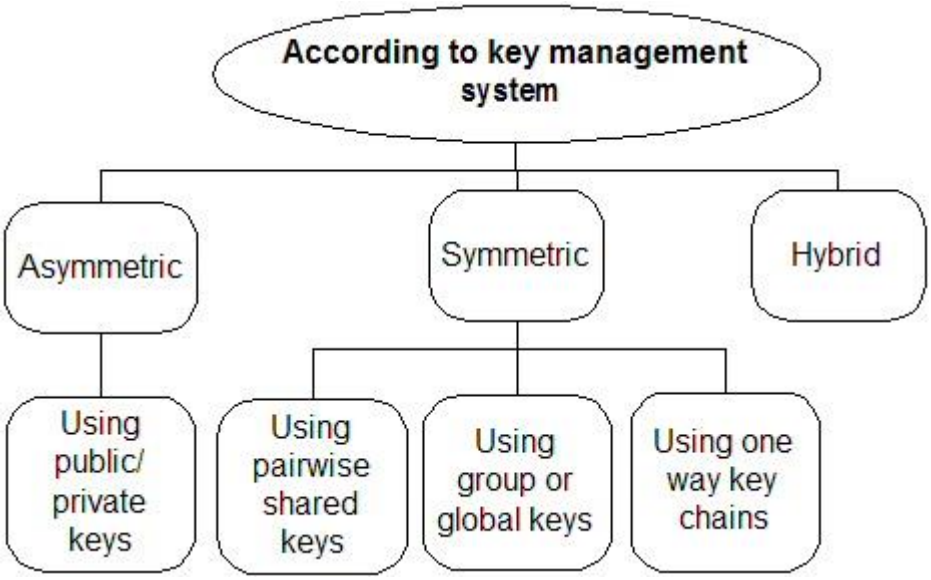


Figure 8.3: Classification of authentication schemes according to key management systems

8.3.1 Symmetric Authentication

In a symmetric key scheme there is a mutually agreed trusted server S which establishes the trust relationship between A and B. Each entity shares a secret key with the server. First, A asks S to establish a relationship to B. Then S sends A and B a session key K which these can use for authentication. A typical scenario can be:

A sends a hello message to S.

S sends $E(K, K_A)$ to A.

S sends $E(K, K_B)$ to B.

For each authentication, do the following:

A computes $MAC_M = MAC(M, K)$ and sends M, MAC_M to B.

B checks whether $MAC_M = MAC(M, K)$.

Here, E(M, K) is the encryption of a message M by the key K, MAC(M, K) is the message authentication code of a message M by a key K, and K_A is the shared key between the server S and A.

The public key schemes provide a signature but the symmetric key schemes provide a key agreement by a central server, and then an authentication process. First three steps only need to be performed once for each pair A and B if the entities store the

shared key K. The remaining steps are performed for each authentication process. However, if all devices share the same key, or if there are predistributed keys first three steps are not necessary :

A computes MAC_M = MAC(M, K) and sends M, MAC_M to B.

B checks whether MAC_M = MAC(M, K).

8.3.2 Asymmetric Authentication

This is a public key scenario where each entity has a certificate (PK) issued by a certificate authority (CA) and an assigned public/private key pair. A message authentication works as :

A signs a message M as S = SIGN(M, SK) and sends M, S, (PK) to B.

B verifies whether (PKi) valid and whether VERIFICATION(S, M, PK) valid.

SIGN(M, SK) is the signature of the message M by the private key SK, and VERIFICATION(S, M, PK) be the verification of the signature S to the message M by the public key PK. VERIFICATION(S, M, PK) is valid if S is the signature of M by the corresponding secret key of PK, if S = SIGN(M, SK) and it is invalid otherwise.

8.3.3 Hybrid Authentication

In hybrid systems the algorithm uses both symmetric and asymmetric algorithms for authentication. In a public key scenario, an equivalent approach is to use a key agreement scheme such as Diffie Hellman followed by a symmetric MAC scheme for each authentication process. A sample hybrid authentication scheme can be as follows:

A sends B its public key K_A and B sends A its public key K_B.

A computes K = a . K_B.

B computes K = b . K_A.

For each authentication, do the following:

A computes MAC_M = MAC(M, K) and sends M, MAC_M to B.

B checks whether $MAC_M = MAC(M, K)$.

In the above scenario, a and b are the private keys, and $K_A = a \cdot G$ and $K_B = b \cdot G$ with base element G are the public keys of A and B, respectively. Computation of K is done like $K = a \cdot K_B = a(bG) = b(aG) = b \cdot K_A$.

8.4 According to Replay Attacks Protection

Authentication algorithms can be classified according to their replay attacks protection mechanisms. Some of the algorithms use timestamp, some of them use sequence numbers and some of them use nonce (a nonce is an unpredictable bit string, usually used to achieve freshness) for replay attacks protection (Figure 8.4).

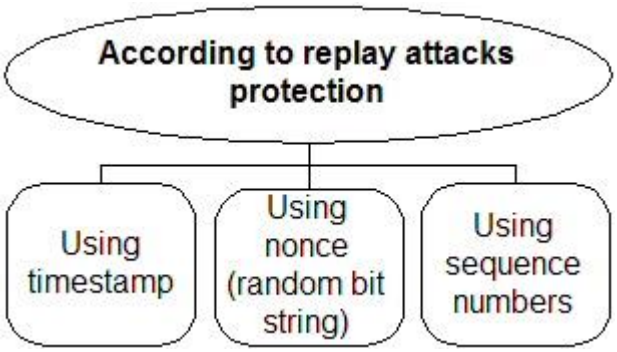


Figure 8.4: Classification of authentication schemes according to replay attacks protection

8.4.1 Timestamp Usage

In some of the protocols a timestamp is added to every packet for checking freshness. It is needed for some type of DoS attacks. This model needs time synchronization.

8.4.2 Sequence Number Usage

In some of the protocols a counter value is added to every packet and the counter value is incremented after a packet is sent.

8.4.3 Nonce Usage

A random number is sent which is known by the receiving party is added to each packet for checking packet freshness.

8.5 According to Involved Parties

Authentication algorithms and frameworks can also be classified by using the involved parties. There can be unilateral, mutual and broadcast authentication types (Figure 8.5).

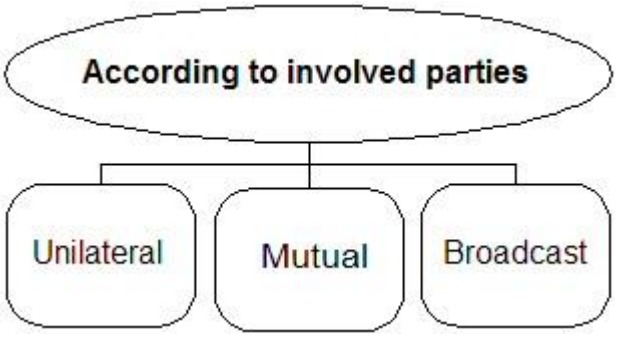


Figure 8.5: Classification of authentication schemes according to involved parties

8.5.1 Unilateral Authentication

Only one party needs to do authentication for message authentication. All of the message authentication protocols only rely on this kind of authentication.

8.5.2 Mutual Authentication

This type is valid for only entity authentication. Mutual authentication can be achieved by performing a unilateral authentication twice. A sample scenario can be:

B sends random r_B to A.

A computes $MAC_A = MAC(r_B \| r_A,K)$ and sends MAC_A, r_A to B.

B checks whether $MAC_A = MAC(r_B \| r_A,K)$, computes

$MAC_B = MAC(r_A \| r_B,K)$ and sends MAC_B to A.

A checks whether $MAC_B = MAC(r_A \| r_B,K)$.

8.5.3 Broadcast Authentication

Broadcast authentication is the process where one entity authenticates messages to several parties. In symmetric schemes MAC usage with pairwise shared keys only

provides mutual (pairwise) authentication whereas a digital signature also provides broadcast authentication.

For providing broadcast authentication with symmetric schemes multiple keys and multiple MACs can be used. The sender can append the MAC calculations for all of the involved parties in its target to the packet.

9. RESEARCH AND ALGORITHMS

9.1 Proposed Algorithms and Specifications

For this thesis nine proposed authentication algorithm and frameworks are studied and they are described in detail. In section 9.2 a distributed authentication model is discussed. The model uses the idea of threshold cryptography. The authentication model suggests using a distributed trust model. This model is mainly based on human behavior. The nodes in the network ask a trust value for a node by using its trusted nodes. For example for an (n, t) threshold scheme if a node asks n nodes for a trust relationship with a specific node and t of the nodes tell that the node is trustworthy, then according to this model the node is said to be authenticated.

In section 9.3 an authentication framework for hierarchical sensor networks is proposed. It is mainly based on Tesla [13]. The main difference between Tesla and this framework is the usage of asymmetric key systems. This framework does not depend on asymmetric systems; the distribution of the keys is done by a system wide private key.

Another lightweight authentication scheme is proposed in section 9.4. This scalable and efficient authentication protocol is called LHAP (A Lightweight Hop-by-Hop Authentication Protocol). This model uses one way key chains for authentication. For the initial trust Tesla algorithm is used. By using the digital signature scheme of Tesla, a secret key between two entities are defined. By using this secret key and some one way hash functions a key chain is created. This algorithm uses message authentication codes and these codes are calculated by using the specific key in the key chain. Normally, there are defined time periods in Tesla where one of the keys in the key chain is selected every time period. The key that is used in message authentication code calculation is disclosed in the next packets at the end of the period. However, this framework does not depend on periodic or delayed key disclosure. It proposes using traffic keys for immediate authentication. There is also a traffic key chain in addition to the Tesla key chain. The traffic keys in the chain are sent to the nodes by using Tesla. Source nodes append the traffic key in the packet and enables immediate authentication.

In section 9.5 another encryption and authentication protocol is proposed. It is called LEAP (Localized Encryption and Authentication Protocol) [15]. It is supposed that the network is cluster based and static. The proposed protocol states that the different types of messages have different security requirements. There is a base station and every node in the network share a secret key with the base station. There is also a cluster key where all the nodes in the same cluster share the same key. Neighboring nodes also share secret keys between each other. It is based on μTESLA [16] which needs delayed key disclosure and loose time synchronization. However this framework needs authentication in every hop and using μTESLA with delayed key disclosure is not suitable, so immediate authentication is supplied by adding an extra MAC calculation with the cluster key.

An efficient and secure authentication protocol is proposed in section 9.6. It is a multicast based system which uses Tesla. For immediate authentication it adds the next packet's MAC in the current packet. For example it appends the third packet's MAC calculation in the second packet. So the receiver can immediately authenticate the third packet when it is received. This algorithm needs time synchronization and for synchronizing the time an asymmetric scheme is used.

In section 9.7 another cluster based authentication scheme is proposed. It is designed for hierarchical networks. Authentication is done by using the special tags and a session key that are generated by the cluster head. There is also a cluster key which is distributed by using asymmetric systems (system public/private key pairs). Every node should have a public/private key pair to be used as a cluster key.

Another authentication scheme is described in section 9.8 where there are 3 types of security levels. "*Security level I is for mobile code which the most sensitive information and security level II is dedicated to the location information in messages, the security level III is applied to the application specific information"* [18]. For this framework RC6 is used with its configurable parameters. There is also a key chain is defined in this algorithm. One of the keys in the key chain is active in every time period. The key that is going to be used is defined by the participation of the nodes in the network. By using this main key the keys for other security levels are defined.

In section 9.9 SPINS [16] authentication and encryption scheme is described. This framework uses SNEP and μTESLA and it is based on broadcast communication. SNEP is an encryption protocol which adds 8 bytes to messages and which uses a counter. This framework uses a key chain and the active key that is used in the MAC calculation is not sent with the next packets as in μTESLA. It is disclosed as a

special packet where this packet is encrypted by using SNEP. Within this framework for SNEP, RC5 and counter mode is used and for MAC calculation CBC-MAC mode is used.

Finally, TinySec [19] which is a link layer authentication and encryption scheme is described. It uses a symmetric key system for authentication and encryption. It uses the same algorithm for both encryption and authentication. It uses message authentication codes in CBC mode. It uses Skipjack algorithm for encryption purposes.

The general properties and classification of the described algorithms can be seen in table 9.1.

Table 9.1: Classification of the Described Algorithms.

Described Algorithm's Section	Immediate Auth.(I), Delayed Auth.(D), Threshold Auth.(T)	Static Topology(S), Dynamic Topology(D), Cluster Based(C), Hierarchical(H)	Symmetric(S), Asymmetric(A), Hybrid(H)	Time Synchronization(T), Nonce Usage(N), Sequence Numbers(S)	Unilateral (U), Mutual(M), Broadcast(B)
9.2	T	S,D	S	-	U
9.3	D	D,H	H	T	B
9.4	I	D	H	T	U
9.5	I	S,C	S	-	U
9.6	I	D	H	T	B
9.7	I	D,C,H	H	S	U
9.8	I	S,D	S	T	B
9.9	D	S,D	S	T	B
10	I	S,D	S	S	B

9.2 A Distributed Light-Weight Authentication Model for Ad-hoc Networks

In this work a security model for low-value transactions in ad-hoc networks is presented. It is focused on authentication, since this is the core requirement for commercial transactions. The proposed model does not require devices with strong processors as public-key systems.

9.2.1 Previous Works

9.2.1.1 Distributed Trust Model

These kinds of systems should be robust against attackers. The Distributed Trust Model [8] mainly depends on the values about trust. In this model there must exist a value of trust describing how much an entity is trusted. There can also be a hierarchy of trust relationships. The Distributed Trust Model protocol exchanges trust-related information for its protocol. The model assumes that trust relationships are unidirectional and take place between two entities. In this model the entities have values for trust relationships. "*The protocol works by requesting a trust value in a trust target and after getting a response an evaluation function is used to obtain an overall trust value in the target*" [9]. The protocol also allows recommendation refreshing and revocation. The recommender sends the same recommendation with another recommendation value to revoke. The model is suited to establish trust relationships that are less formal, temporary or targeting ad-hoc commercial transactions.

An implementation of Distributed Trust Model for ad-hoc networks would be particularly vulnerable to malicious and compromised agents. Since ad-hoc network topology is flexible it is unclear if enough trustworthy entities are available to obtain a recommendation. Finally the requestor could ask for information regarding what other entities think about each other. This could be data that other entities would not want to reveal.

9.2.1.2 Password-Based Key Agreement

Suppose there is a group of people who want to set up a secure session in a room without any support infrastructure. The properties for such a protocol are stated as:

Session Key [9]: From the Initial password known by those entities the session key can be fetched.

Contributory key agreement [9]: the session key is formed of contributions from all entities. This ensures that if only one entity chooses its contribution key randomly all other entities will not be able to make the key space smaller.

Tolerance to disruption attempts [9]: the protocol must be protected from unwanted messages. It is also assumed that the possibility of modifying or deleting messages in such an ad-hoc network is very unlikely. A weak password is sent to the members of the group. Upon receipt, each member contributes to part of the key and signs this data by using the weak password. Finally a secure session key for setting up a secure channel is derived without any central trust authority or support infrastructure.

This model is widely used and is well suited for small groups. Authentication is done outside the system. The group members authenticate themselves by showing their passports or by agreeing on a common knowledge. The model is not appropriate for more complicated environments. The problem arises for large groups at different locations.

For this kind of systems there is not an available secure channel to distribute the initial password. At this point it seems that for setting up a secure channel an existing support infrastructure is required.

9.2.1.3 Resurrecting Duckling Security Policy

This policy is appropriate for weak embedded systems which can not make public key operations. It depends on a master-slave relationship. In this model master and slave share a common secret and transient because only the master is able to solve the association. The proposed solution is called Resurrecting Duckling Model [10].

"*The duckling is here the slave device while the mother duck is the master controller. The duckling will recognize the first entity that sends it a secret key on a secure channel which may be by physical contact as its mother. This procedure is called imprinting*" [10]. The duckling will always obey its mother. The mother has an access control list and tells the duckling to whom it can talk. The connection between mother and duckling is broken by death and after that the duckling accepts another imprinting.

Death may be caused by the mother itself, a timeout or any specific event. The whole security chain is a tree topology of master-slave relationships. The root of the tree can be a human being. The root controls all devices and every node controls all devices in its sub trees. In this policy if one relationship is broken the relationship of the whole sub tree is also broken. This security model can support large ad-hoc

networks. A possible scenario is a battlefield of smart dust soldiers (acting as slaves or siblings) and their general (acting as the master).

The Resurrecting Duckling scheme is a suitable model for a hierarchy of trust relationships [9]. It particularly suits cheap devices that do not have a processor for performing public-key operations. This perfectly works for a set of home devices, for instance.

For a battlefield the soldiers are siblings and obey their mother, the general. If one soldier wants to authenticate to another device it has to present its credentials. The second device can then check the credentials by using its policy. But if not all soldiers use the same credentials, for instance the same secret key to prevent it to be stolen by the enemy, this model can not operate. If all devices use the same key then the other side can do some physical attack to recover the key because it would compromise all nodes. Since the devices cannot hold a list of all valid credentials it seems that this method can not be used [9].

9.2.1.4 Distributed Public-Key Management

Key management for public-key systems requires a centralized trusted entity called Certificate Authority (CA) [9]. The CA gives certificates by binding a public key to a node's identity. One constraint of this management system is that the CA should always be available because certificates might be renewed or revoked. This model can be improved by the other CA's availability.

By using threshold cryptography, distributing trust to a set of nodes and letting them share the key management service can be implemented. For (n, t + 1) threshold cryptography scheme n parties share the ability to perform a cryptographic operation so that any t + 1 party can perform this operation. But t parties can not perform this operation [9].

Using this scheme the private key k of the CA is divided into n shares (s_1, s_2, . . . , s_n), each share being assigned to each special node. Using these shares a set of t + 1 special nodes are able to generate a valid certificate. Even if compromised nodes deliver incorrect data the service is able to sign certificates. "*Threshold cryptography can be applied to well used signature schemes like the Digital Signature Standard (DSS)*" [11]. In another approach the system replaces the centralized CA by certificate chains. Users issue certificates if they are confident about the identity or the given public key belongs to a given user. Each user stores a list of certificates in its own repository. To obtain the certificate of another entity the requester builds a

certificate chain using his repository list and implicitly trusted entity's lists until he has found a path to an entity that has the desired certificate in its repository.

The Threshold Key Management system is a way to distribute a public-key system. Public-key systems support a secure framework for high-value transactions. Two entities that do not have any common knowledge can trust a CA to substitute this knowledge. When an entity wants to prove its identity it goes there with its public key and shows its passport to CA. The CA proves the identity and then binds this identity to the public key and signs the entity's certificate. If the CA signs certificates without proving the identity the model cannot be used for high-value transactions.

For this infrastructure, for obtaining a certificate going through the certificate chain high computing power and computation time is required and also each node should perform public-key operations. Among these operations are: checking the received certificate for authentication (signature verification) and signing it (signature generation). These operations are done in a sequential manner. It can be said that a central CA can be used for applications with high-security demand and to ensure availability the CA can be replicated.

9.2.2 Distributed Light-Weight Authentication Model

For specific situations users can find the most appropriate system for their applications. For example, for a meeting in a room the password based key exchange method can be used, for a network defined by a hierarchy of trust relationships the resurrecting duckling policy is the best method. It can be said that none of these models is the best solution for low-value transactions in wireless sensor networks.

Their proposed model is strongly based on human behavior. Here the human society can be seen as an ad-hoc network. They used the Distributed Trust Model [8] to establish trust relationships and extended it by a request for references. Each entity in the network stores a table of trustworthy entities from which paths between two entities can be found by using the tables of other entities. This type of algorithm is used for a self-organizing public-key system. The algorithm represents the threshold cryptography. In this type the compromised nodes can not harm the result of the operations if their number is below a threshold.

"*The main purpose of the algorithm is not to make transactions really secure. Actually the main goal is to make the attacker mostly get authenticated falsely*" [9]. The node asks some common knowledge about a node to other nodes in the system. If no common knowledge is found a trust relationship can be derived using a trusted

third party. The CA or the mother duck can be used as a trusted third party according to the used algorithms (public key systems or hierarchical relationships). This model supports cooperation and feedback [9].

9.2.2.1 Description of the Proposed Model

If Alice wants to verify Bob's identity, Alice starts asking Bob about common knowledge. This can be a secret key or a common knowledge about a recent transaction. Bob can prove his identity if there is some common knowledge. Otherwise, Alice starts asking to nodes taken from her list of trustworthy entities. In case possible return values can be 'yes' or 'no' to say it is Bob or not. Suppose Alice asks Cathy about Bob, she should first check Cathy's identity by asking her about common knowledge (assuming that this transaction was encrypted) then Cathy looks if Bob is on her list of trustworthy people and checks his identity, or forwards Alice's request in the same manner until an entity is found that knows Bob and can prove his identity. Once found the information is sent back to Alice, including all entities in the recommendation chain.

By using another algorithm Alice can ask about Bob's identity to random entities. She might also ask Cathy and all other entities in the recommendation chain to do the same or Alice can ask Bob to give the devices that he has done transactions with recently. Alice can then ask these devices if they know Bob. Since Bob could easily set up the entities he gives as references, Alice can ask again her network of trustworthy entities if they know the references. A good reference would be an entity that has a relationship with Alice and Bob. When a link between Alice's trusted network and Bob's reference network is found a direct relationship between Alice and Bob can be setup. In this short chain the requests are sent out recursively [9].

Using both requests for recommendations and references the two trust networks are browsed starting at Alice and Bob until a connection is found. This keeps the number of involved nodes low. After Alice has received the results of her request she has to evaluate the data.

In the evaluation phase Alice decides if Bob is trustworthy or not. For more detailed systems the function might output an upper limit for a transaction value when dealing with Bob instead of "yes" or "no" answer. For data evaluation Alice can check the recommendation chains for suspicious nodes and can ignore these chains if necessary. For the evaluation function the algorithm brings the idea of threshold cryptography. If Alice receives n answers to her request, then the algorithm wants

that the result is not influenced by t or less malicious agents, where n is considerably larger than 2t [9].

For an (n, t) scheme, the algorithm requires that exactly n nodes respond and t/n*100% or less compromised nodes that respond cannot affect the result. A very suspicious person could configure its device such a way that it requires only "yes" answers while less suspicious users would allow some errors.

Following an authentication of an entity, a secure channel can be initiated. A perfectly secure channel cannot be set up if there is no common knowledge between both entities. If encryption is going to be used Alice can send Bob a secret over the trustworthy path which is used to derive a relationship. Bob then sends back another secret using a random path. As a result Alice and Bob can derive a shared secret key by combining these secrets. Within this infrastructure only trustworthy entities can obtain the secret key by eavesdropping Bob's answer.

For this kind of infrastructure the entity should participate in most of the requests and use its battery power and the devices that give recommendations can also expect to receive answers to their requests. Devices that never answer to any request will be removed from the list of trustworthy nodes or can be put on a list of untrustworthy devices [9]. This can be a simple punishment method. Furthermore entities might receive rewards for good recommendations. Finally a feedback method is proposed to make the system more robust. If Alice received from Dan via Cathy a positive recommendation about Bob's identity, but then gets cheated by Bob, she can inform Cathy and Dan about their wrong recommendation. Also she can put Cathy and Bob on a list of untrustworthy devices. The reward and feedback methods give some quality and responsibility function to the algorithm.

9.2.3 Security Analysis

"*The main model is not based on pure mathematical foundations and leaves the individual security policy open*" [9]. For example Alice wants to authenticate a device which tells that it is Bob but it is actually Cathy. By faking references Cathy can claim that she is Bob. If Alice asks these references for another reference the number of references she had to fake rises exponentially.

It is stated that first Cathy had to compromise or fake nodes from Alice's list of trustworthy entities. Finally she also had to compromise or set up enough random nodes such that the probability that Alice exactly asks these nodes is high enough.

The probability is configurable by Alice by setting the threshold value (n, t) in her local security policy.

To enhance the channel's security they replaced a shared knowledge or trusted third party by a path in the network of trusted entities. More efficient algorithms should be studied and found in this area. Feedback system is another issue which has to be efficient and should detect fakers quickly, and take appropriate action to prevent repeated attacks [9].

9.2.4 Conclusions

This model's main idea is the human behavior and it is suitable for the ad-hoc networks. It uses recommendations and references to learn and check trust relationships. A cooperation and feedback system makes the system more robust and more qualified. The hardware requirements for the device are low. Using this model, nodes in an ad-hoc network can set up a secure channel.

9.3 Authentication Framework for Hierarchical Sensor Networks Using Tesla

In this model data and entity authentication for hierarchical ad hoc sensor networks is provided. The sensor network consists of three tiers of devices with different levels of computational and communication capabilities. The lowest tier consists of memory and processing power constrained sensors that are unable to perform public key cryptography. Tesla certificates are used for this type of devices.

"*The framework authenticates incoming nodes, maintains trust relationships during topology changes through an efficient handoff scheme, and provides data origin authentication for sensor data. The framework assigns authentication tasks to nodes according to their computational resources*" [12].

As it is known that, ad hoc networks have capacity constraints and to overcome this problem, this algorithm proposes a hierarchical ad hoc network. The model focuses on the security conditions of hierarchical ad hoc sensor networks for performing authentication and compares it with flat ad hoc networks. In this work a three tier hierarchical ad hoc network is proposed and an authentication framework for that three tier hierarchical sensor network is developed.

9.3.1 Hierarchical Sensor Network

It is stated that some theoretical studies showed the limitations of the flat topology and so as an alternative to flat ad hoc topologies, hierarchical ad hoc networks have been proposed in recent years. Hierarchical approach has better performance than flat ad hoc networks according to the initial analysis. In this work it is supposed that most sensor data are destined for the Internet and using hierarchical approach number of hops to reach destination is really reduced.

There are three classes of wireless devices which can be seen on figure (Figure 9.1) [12]. There are high power access points that route packets received via radio links to the wired infrastructure (B), mobile medium powered forwarding nodes that relay information from sensor nodes to access points (C), and low powered mobile sensor nodes that have limited computing capability (D). There is an Internet-based application (A) that is connected to the sensor network through the access points.

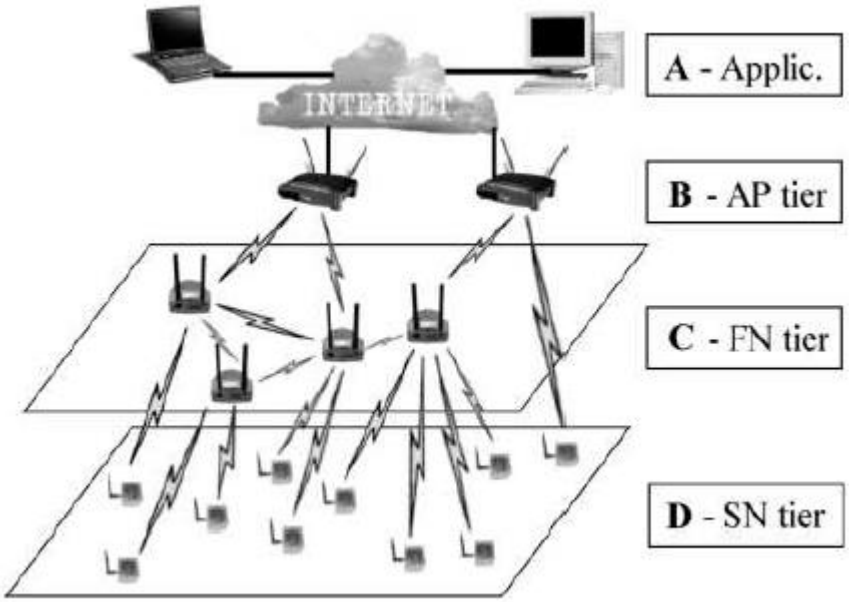


Figure 9.1: Three tier hierarchical ad hoc sensor network [12].

There are different levels of computational power within the sensor networks. Ordinary ad hoc sensor networks assume that all nodes have the same computational capability. In this work the three-tier sensor network architecture consists of three types of devices with different capabilities.

It is supposed that sensors do not communicate with each other and the main idea of a sensor is to feed data to the application. The application is responsible for making decisions based on the results it receives. Sensor nodes only route their packets via

the higher-level nodes and they don’t communicate and authenticate within each other.

In this framework there exists forwarding nodes and the purpose of the forwarding nodes is to relay messages from the sensors to the access points. Forwarding nodes have two wireless interfaces, one that communicates with sensor nodes, and one that communicates with access points. They do not necessarily perform measurements themselves [12].

9.3.2 Tesla and Tesla Certificates

As we know, the most widely used certification systems are based on public key cryptography, but the sensors have resource constraints and should not have to use public key cryptography. Within this protocol a certificate structure that does not employ public key cryptography is offered. A certificate based authentication system for low-powered devices is proposed.

Tesla [13] is a broadcast authentication protocol that use symmetric cryptographic functions. Tesla enables low-powered nodes to perform source authentication. The proposed framework uses Tesla in order to authenticate the devices in the network by several ways. As opposed to the Tesla the clients in this model are not using the public key infrastructure. The application encrypts the initial Tesla key with the appropriate shared key and then sends the key to each of the network nodes [12].

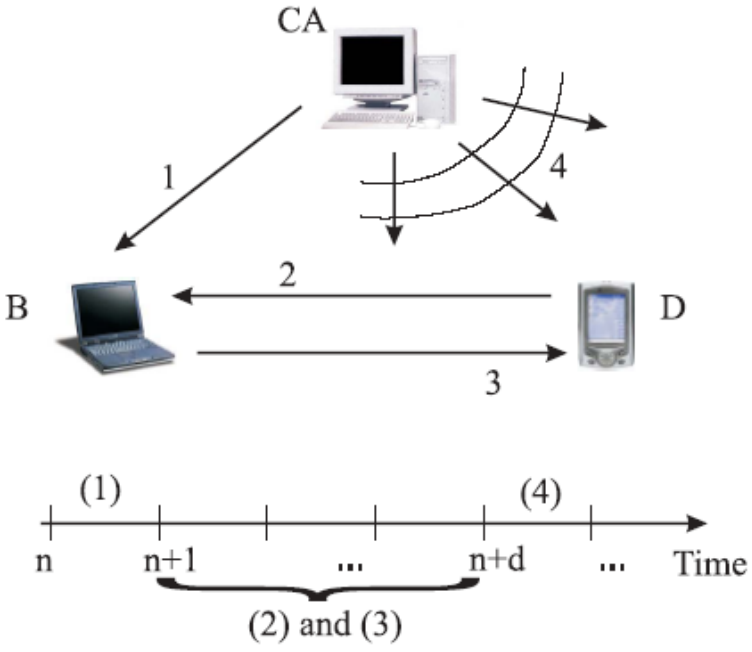


Figure 9.2: The steps involved in using Tesla certificates [12].

Within this framework there is a certificate authority (CA) who is responsible for creating the certificates. A sample scenario is given for the steps involved in Tesla certificates. CA is going to generate certificates for entity B (Figure 9.2). D is a low powered device and will use B's service.

First the CA generates Tesla certificates periodically for the entity B which is labeled as “1” in figure 9.2. During time slot n, the certificate authority (CA) doesn't sign the Tesla-certificate with its private key, but uses the non-disclosed Tesla key to create a MAC that is included in the certificate. B's public key is replaced its authentication key, which is encrypted by the CA using the Tesla key. There is also TSA which is a timestamp addressing the certificate's expiration date. The certificate is sent to B along with the matching authentication key.

After that, sometime between time n and n + d, D contacts B requesting to use B's service (Labeled as “2” in figure 9.2). After the request step B must prove its identity to node D. B sends an authentication packet, which consists of the Tesla certificate and a MAC that was created using B's authentication key (Labeled as “3” in figure 9.2). After receiving the authentication packet, D checks the freshness of the certificate by checking the timestamp of certificate to make sure that it arrived before time n + d. At time n + d the CA announces the key. If Cert is fresh, D buffers the authentication packet.

Later, the CA announces its Tesla key at time n + d (Labeled as “4” in figure 9.2). When D receives the key it checks the authenticity of the Tesla certificate by checking MAC, then it decrypts B's authentication key and checks MAC. If D receives the Tesla certificate Cert before the CA announces the Tesla key, D is able to check the identity of B.

The lifetime of a Tesla-certificate depends on the announcement time of the Tesla key that the CA used when creating the MAC and encrypting the authentication key of B. It is stated that the lifetime of a Tesla-certificate is short. If the key disclosure time is short then the delay in the authentication process at node D is also short. It is seen that this results in increased overhead when issuing new certificates [12].

9.3.3 Overview of the Authentication Framework

The main goal of this framework is to ensure that the data received by the application (A) is sent by the sensor node (D) and to verify that the data hasn't been modified on its way to the application. For these purposes there exists an authentication service. By this service the incoming nodes are authenticated and shared secrets among the

nodes and the application are distributed. It also keeps the changes in the network topology and provides data origin authentication for sensor node data.

In this framework an initial trust is needed. When an incoming node wants to join the network, it should supply some trustworthy information. The trustworthy information should be some data confirmed by a trusted entity. Therefore it is stated that each node that wants to join the network must have a personal initial certificate (iCert) that is issued by the network's trusted third party (TTP).

The TTP is a node that is able to perform RSA signatures and which is trusted by all nodes. There exists a public key which is known by all nodes of the network that are able to verify RSA signatures which the TTP uses. For joining the network, a node or access point should present its iCert. This iCert should be checked by the application A. If the iCert is valid, A will establish a shared secret with the node or the access point whichever wants to join the network. After that the new node can access the network and communicate with the application [12].

It is stated that for this framework there is a need for authentication which is done between the nodes. This method should be fast enough to be used by the computationally weak sensor nodes. The application A enables them by periodically broadcasting runtime certificates (cert) for each access point and sensor node of the network.

For a sample scenario suppose that a sensor node D that wants to send a packet. First it creates a MAC using the secret key it shares with the application and appends it to the data. The application uses the MAC to verify the data's origin. Then if D wants access to the Internet, it creates another MAC using the secret key it shares with its gateway access point B. The access point will use the MAC to perform access control and it checks whether D has rights to send data to network.

9.3.4 Certificates

Certificates are the major tool within this framework. It builds the main entity authentication service. By using certificates the nodes that don't share a secret key can establish a trust relationship. The proposed framework has two types of certificates which are initial and runtime certificates [12]. Each node needs an initial certificate to join the network. The runtime certificates are periodically issued by the application, which are going to be used within the network's lifetime.

Each access point or sensor node must own a certificate issued by the network's trusted third party. They assumed that the access point B is a device of high computational power and battery resources. So within the framework access points validate and perform RSA signatures. Each access point has an RSA public and private key and a certificate issued by the Trusted Third Party.

The sensor node D gets its initial certificate from the trusted third party if it wants to communicate with a certain application A. The TTP gives an iCert to the sensor node D with a unique key. D uses this key to authenticate itself against application A. For the iCert security D's initialization key is encrypted with A's public key and only A is able to obtain the key from the certificate and proof D's authenticity.

The main purpose of runtime certificates is to keep on the trusted communication between the initial authenticated nodes during the networks' lifetime. The runtime certificates use the trust relationships between the application and the nodes of the network to create new trust relationships among them.

It is stated that each sensor node D should access the access points' runtime certificates, so RSA-based certificates cannot be used. Instead of using RSA based certificates Tesla certificates are used. The usage of MAC in the algorithm proves that this certificate was issued by the application [12].

9.3.5 Certificate Renewal

As in every ad-hoc network, during the lifetime of the network, trust relationships change. Misbehaving nodes have to be identified and must not be allowed to remain connected to the network. So certificate renewal is needed.

The framework states that the application issues a new certificate for each connected access point B after a certain period of time. In the beginning of time slot n, A sends to B the new certificate with B's authentication key, which will prove B's identity to a sensor node D during handoff and a MAC. Checking MAC, B can verify that this certificate was issued by the application A.

The application issues a new certificate with the authentication key that D uses for sensor node handoff and a MAC for each connected sensor node D after a certain period of time. This period of time can be much greater than the Tesla time slots as the sensor node runtime certificates don't depend on the application's Tesla keys [12].

9.3.6 Entity Authentication

9.3.6.1 Access Point

In this framework the access point B isn't a normal mobile device. It supplies a wired connection to the Internet on a specific place for sensor nodes. It is stated that an authentication process is needed for the access point because it will provide access control between the application and the sensors. If B knows the address of A and A's public key, it will contact its application A and send a service offer. B signs the offer with its private key and appends its certificate before sending the offer. To accept the offer, A checks the signature with help of the certificate. If the signature cannot be verified it sends a signed lack of interest (LoI) message.

It is told that if the verification is successful it returns an accept-message including a shared secret key and the 'access point group key' (that will be used in the sensor node handoff scenario), encrypted with the access point's public key and signed with its private key. The application access point authenticity and the shared secret key are the basis for authenticity of the entire network [12].

9.3.6.2 Forwarding Nodes

As it can be seen forwarding nodes are mobile devices and they move in the entire network. In this framework forwarding nodes have two wireless network interfaces and forward data packets sent by sensor nodes. The packets that can't reach the access point directly are sent through the forwarding nodes. In the proposed framework, forwarding nodes only authenticate themselves in the assured mode [12].

9.3.6.3 Sensor Nodes

In this framework the sensor nodes are highly mobile and have limited computational capabilities so the authentication and encryption algorithms should be light weight. When a new sensor node D enters the network, it sends a request to the application. There can be a node C that simply forwards the request of the sensor node. When the sensor node authentication request is received by an access point B, the ID of B and a MAC that is created using the key of B that it shares with the application A is attached to the request. Once A gets the request, it checks the certificate and creates a shared secret between the access point B and the sensor node D by returning two instances of a new key, one encrypted with D's initialization key and the other encrypted with the key it shares with the access point. A also establishes a shared secret with the sensor node by appending another new key, which it also encrypts

using D's initialization key. It is seen that after receiving A's answer, D shares a secret with its gateway access point B and the application A [12].

9.3.7 Data Origin Authentication

The framework's data origin authentication service enables the application A to check whether the received data are sent by a valid sensor node. The sensor node D uses the secret key that it shares with the application to create a MAC that it appends to the data packet. The application authenticates data by appending a MAC created with the secret key it shares with the relevant node.

It gains access to the wireless part of the network by appending another MAC that it creates using the key it shares with the relevant access point. Access points only forward authenticated application data to avoid some attacks against the wireless devices. Broadcast application data is authenticated using the Tesla protocol.

9.3.7.1 Sending Sensor Data in Weak Mode

As it is known that there are some changes in the topology and the sensor node D doesn't know if its data will arrive directly at the gateway access point B or will be forwarded by forwarding node C that forwards it either to another forwarding node C or to B. According to the framework the format of the data packets depends on the sensor node D, the gateway access point B and the application A.

It is stated that the packet contains three fields in addition to the data. These are two MAC fields and one encrypted random number field. If the forwarding node C receives the packet, C forwards it without doing any modification to the packet. Once the gateway access point B gets it, B checks the first MAC. If it is valid, B removes the MAC and the random number from the packet, adds a new MAC using the secret key it shares with the application and sends it on. After that, it decrypts the random number, adds one, encrypts the result and sends it back to D. When D receives the result it can be sure that the data is on the right way to the application. However, with this mechanism D can not figure out who delivered the packet to the gateway access point [12].

"By this way a misbehaving forwarding node can copy the packet without being detected. A sensor node that wants all nodes on the path to the application to be authenticated must choose the assured mode" [12]. When the application A receives the packet, it checks both MACs and, if both are valid, processes the data. If one of them is invalid, A returns a data-reject-message (dRej). This message includes

information about which MAC is invalid. It appends two MACs that enable B and D to verify the application as the sender of the reject-message.

9.3.7.2 Sending Sensor Data in Assured Mode

It is stated that the assured mode provides authenticity along the path of the packet. This causes some additional message exchange, higher computational overhead and less flexibility. A sensor node D that wants to send in assured mode first sends an assured data-request (asdReq) that contains an encrypted secret key that will be used to install a shared secret between D and the forwarding nodes along the path. If there is a forwarding node C between D and B, C will sign the packet and append its cert before forwarding it to B [12].

The gateway access point B that gets the packet first checks the certificate, then the signature. If they are valid B replies with an assured data-confirmation (asdConf) that includes the secret key encrypted with the forwarding node's public key. The forwarding node extracts the secret key from the confirmation packet and uses it to create and append a MAC to the confirmation message before sending it on to the sensor node. Once the packet reaches the sensor node, it will check the MAC and if the MAC is valid, D can be sure that its gateway access point trusts the forwarding node. To make sure that the data takes the authenticated path, the sensor node additionally encrypts the random number with the secret key.

In the case that more than one forwarding node lies on the path between the sensor node and its gateway access point, each of them appends its signature to the request before forwarding it in the direction of the access point. It is stated that the access point will establish the additional needed shared secrets between sensor and forwarding nodes. D will successively encrypt the random number using each of the keys in the appropriate sequence before sending data.

9.3.8 Security Analysis

The use of message authentication codes in the framework protects all data against malicious modifications and information forgery. The access point restricts battery consumption attacks by making it impossible to launch such attacks from outside the wireless part of the network.

Since all initial authentication is done by the application that is the master of the network, compromised sensor nodes can't make any damage to the network other than feeding the application with wrong data. Because of the constrained battery

resources, denial of service (DoS) attacks launched by compromised sensor nodes are unlikely to happen. Even in the case of such an attack, the application can easily find the origin of the DoS packets and end the sensor node's trust relationships in the entire network.

Since the sensor data packets include a challenge-response mechanism, false forwarding of packets or the deletion of packets by a forwarding node will be detected by the sensor node. The framework doesn't provide the possibility to avoid or detect the malicious duplication and distribution of sensor node packets.

According to this framework if a compromised forwarding node stops forwarding data to or from the sensor node or continuously modifies sensor node data that then gets rejected at the access point, the sensor node must find a different network node to connect to [12].

It can be seen that once an attacker manages to compromise the application, the authentication framework fails. However, if the application itself is compromised there is no use in protecting the sensor devices or data.

9.3.9 Performance Analysis

As the topology changes frequently, the proposed security solutions must be highly adaptable. As the ad hoc network can consist of thousands of sensor nodes, the proposed security mechanisms should be scalable. The proposed architecture is capable of adapting to meet the authentication needs of the sensors resulting from topology changes.

According to the framework the amount of memory required by sensors to store their authentication keys remains the same as the number of sensors increases. However the amount of keys that must be stored by access points and the application will increase as the number of sensors increases, but these devices usually have more memory.

For some performance tests the authors tested 4096-bit message authentication using SHA-1, and 2048-bit RSA signing on a Pentium-4 2GHz Linux machine. They measured that the RSA operation requires roughly 4900 times more power than performing SHA-1. Due to their use of Tesla certificates, the sensor nodes use computationally efficient MACs to perform authentication [12].

9.3.10 Conclusion

"*The framework authenticates incoming nodes, maintains trust relationships during topology changes through a flexible handoff scheme, and provides data origin authentication for sensor data*" [12].

As the framework is designed for hierarchical networks where nodes are treated according to their resource limitations, the sensor nodes don't struggle with the creation or validation of public key signatures. Sensor nodes only perform runtime authentication by Tesla certificates which don't use public key structure.

9.4 A Lightweight Hop-by-Hop Authentication Protocol for Ad-Hoc Networks

9.4.1 Introduction

We know that in ad hoc networks, there are no base stations and each mobile node acts as both a router and a host which are communicating with each other at every time. Currently in most of the wireless sensor networks, network access control is not implemented. Because of the lack of this feature these networks are open to resource consumption attacks. As mobile hosts are usually battery powered, they are open to battery exhaustion attacks.

For instance it is stated that most routing protocols send a route request packet which is broadcasted to all nodes and in these protocols, a node trusts that its neighbors will forward packets for it. It also assumes that the received packets are authentic. The solution to this kind of problem is to provide network access control by authenticating all packets and forwarding only the packets from authorized nodes. If verification fails the packet is immediately dropped. For this proposition to work, a network-wide key shared by all nodes can be used where every node uses it to compute message authentication codes (MACs) on the packets it sends. The received packets can be verified by using the shared key.

This scheme has some disadvantages. For example someone knowing the global key can make the system halt and after that time it is very difficult to identify the compromised node. Also when the global key is known by outsiders, generating a new key and distributing it to all the safe nodes securely is really difficult.

Another authentication mechanism of this kind of protocols is using public key or asymmetric key cryptography. However these techniques usually do not suit well to

ad hoc networks because of the resources of a node like power, computational capacity and bandwidth constraints. Because of these reasons a lightweight authentication protocol for the resource constrained environments is proposed.

LHAP [14] is proposed as a scalable and efficient authentication protocol for ad hoc networks which prevents resource consumption attacks and implements lightweight hop-by-hop authentication. Within this framework the intermediate nodes authenticate all the packets they receive before forwarding them and when a node want to join the ad hoc network, it only needs to perform some authentication operations to build a trust relationship with its neighbors. It is stated that LHAP is independent of the routing protocols which is residing between the data link layer and the network layer.

9.4.2 Main Components of the Protocol

LHAP uses one-way key chains for traffic authentication and TESLA [13] for bootstrapping trust.

9.4.2.1 One-way Hash Chains

A one-way key chain is a chain of keys generated by applying a one-way hash function on a random number repeatedly. For instance, if a node wants to generate a key chain of size N, it first randomly chooses a key, K(N), then computes

$$K(N - 1) = F(K(N)), \quad K(N - 2) = F(K(N - 1)), \ldots, \tag{8.1}$$

repeatedly until it gets K(Q) = F(K(l)). F is a random function which is one-way and with a given K(i), anybody can compute K(i - l), K(i - 2), . . . ,K(0), but they can not compute any keys in K(i + 1), K(i + 2), ...,K(N).

It is stated that for authentication the sender signs the last value in the chain K(0), with its private key and anybody who knows its public key can verify the signature and the authenticity of K(0). Then the sender discloses keys in the chain in an order reverse to that of its generation. A receiver can authenticate K(j) by verifying the equation 8.4 if it has K(j — 1).

$$K(j - 1) = F(K(j)) \tag{8.2}$$

Furthermore, if a receiver did not receive K(j - 1) and the last key it authenticated is K(i), where i < j - 1, it can still authenticate K(j) by verifying the equation 8.5. This property is useful for the toleration of packet losses [14].

$$K(i) = F_{j-i}(K(j)) \quad \textbf{(8.3)}$$

9.4.2.2 Tesla

TESLA [13] is a broadcast authentication scheme proposed by Perrig A. and the main technique used in TESLA is a one-way key chain with a property of delayed key disclosure. After building an authentic key from a one-way key chain between the sender and its receivers using digital signature, TESLA uses MACs for authentications. "*Within TESLA, a sender uses a key K from its key chain as the MAC key to compute a MAC over packet P(i) which is attached to P(i). The key K is stored in the next packet P(i + 1), which allows the receivers to verify the authenticity of K and hence the MAC of P(i). If both K and the MAC are correct, and if the packet P(i) is received before P(i + 1) it can be said that P(i) is authentic*" [14].

So it can be seen that the security condition in TESLA is the sending time of each packet. TESLA needs periodical key disclosure and loose time synchronization where time is divided into many equal length intervals [13]. In each time interval the sender discloses one key from its key chain. For example, if the start time is s, the time interval is T, the sender can publish K(i) at time s + i * T. The "loosely" means that there is an upper bound of the synchronization error between any two nodes. By using this synchronization error and the sender's MAC key disclosure interval, the receiver determines whether the MAC key for a packet it just received has been revealed. If the MAC key has been revealed, the packet is dropped, otherwise the MAC key in this packet is used to verify an earlier packet, and then this packet is buffered until the next packet arrives. One disadvantage of the TESLA protocol is late authentication of the packets [14].

9.4.3 A Lightweight Hop-by-hop Authentication Protocol

9.4.3.1 Assumptions

For this framework it is assumed that the links are bidirectional within the wireless network which means that the nodes use omni-directional antennas and have similar power levels. Another assumption is that if node A sends a packet to its neighbor B, then for another node P hearing, modifying and re-sending it to node B before node

B receives the original packet is not possible if node B did not drop the original packet.

Another assumption is that each node has a public key certificate signed by a trusted certificate authority (CA) and also an authentic public key of the CA. The proposed protocol relies on these public keys to bootstrap trust in the network. As public key certificates which are flexible and scalable are used, a node can authenticate other nodes independently without need for to have pre-shared keys as in symmetric cryptography. And for the TESLA protocol loose time synchronization in the ad hoc network is used [14].

9.4.3.2 LHAP Overview

As the purpose of the protocol is to prevent unauthorized nodes from being able to inject traffic into the network, every node in the network authenticates every packet it receives from its neighbors before forwarding it. A packet which needs multiple hops to reach its destination gets authenticated by each node on its path which is called as hop-by-hop authentication.

The LHAP uses two techniques for the authentication protocol. The first one is lightweight packet authentication, and the second one is lightweight trust management. It is stated that all packets are authenticated on every node on their paths, thus the authentication technique used by LHAP should be as inexpensive as possible which is based on the use of one-way hash chains. To reduce the number of public key operations for bootstrapping trust between nodes, LHAP uses TESLA [14].

9.4.3.3 Lightweight Traffic Authentication

The LHAP does not use periodic and delayed key disclosure as it is not appropriate because a packet should not be delayed at each node in the path from the source to the destination. For example if a packet needs ten hops to reach its destination and the key disclosure period in TESLA is one second, then it will take at least ten seconds to reach to the destination and each node should buffer these packets until they are authenticated. This usage will need large storage requirements at every node.

For the proposed framework the source node generates a one-way key chain that will be used for authentication by its immediate neighbors. The term TRAFFIC key which refers to the keys in one-way key chain is used by this framework. “*When a*

node first establishes a trust relationship with another node, it will obtain an authentic key in this TRAFFIC chain where a source node sending a packet will append a new TRAFFIC key to the packet. All the nodes on the packet's path to the destination receive this packet and verify its authenticity by checking the validity of the attached TRAFFIC key" [14].

9.4.3.4 Trust Management

A public key based technique is used to exchange authentic TRAFFIC keys and provide nodes' trust relationship. For instance a node can sign its most recent TRAFFIC key and send it to every new neighbor, but this approach is not well suited for resource-constrained nodes. LHAP uses TESLA to reduce the number of signature operations and to be suitable for resource constrained environments. It is stated that in LHAP digital signatures are used to bootstrap a TESLA key chain which are then used to provide authentic TRAFFIC keys. In this framework a node authenticates TRAFFIC keys by its TESLA keys and broadcasts them periodically which are called KEYUPDATE messages, to maintain the trust relationship.

When a node does not receive a valid KEYUPDATE message from a neighbor within a TESLA interval, it terminates its trust of this neighbor and they will need to run the trust bootstrapping process to reestablish a trust relationship again [14].

A, B are principals, the identities of mobile nodes. Cert A is node A's public-key certificate issued by a trusted CA. $Sign_A(M)$ denotes the digital signature of message M, signed with node A's private key. M1|M2 denotes the concatenation of message Ml and M2. MAC(K, M) denotes the computation of MAC over message M with key K. $K^T_A(i)$ denotes node A's i'th key in its TESLA key chain, while $K^F_A(i)$ denotes its i'th key in its TRAFFIC key chain.

9.4.4 LHAP in Detail

LHAP consists of two security building blocks: traffic authentication and trust management. Trust management includes trust bootstrapping, trust maintenance and trust termination.

9.4.4.1 Traffic Authentication

The TRAFFIC key chain is used for authenticating the traffic packets that are created or forwarded. For example, if a node A wants to broadcast a packet M then it will send M, K^F_A (i) where K^F_A (i) is the next TRAFFIC key.

Every receiving node verifies the authenticity of this packet by verifying the TRAFFIC key K^F_A (i), based on the most recent TRAFFIC key. If it knows K^F_A (j), j < i, that it received from node A then it verifies the packet and replaces K^F_A (j) with K^F_A (i) in its record. It is stated that in LHAP a node only authenticates traffic packets from its direct neighbors to make replay attacks difficult. "*The TRAFFIC keys enable instant verification of packets and remove the need of disclosing TRAFFIC keys periodically. In LHAP the rate at which a node consumes its TRAFFIC keys can be adapted to the actual traffic rate*" [14].

9.4.4.2 Trust Bootstrapping

When a node wants to join an ad hoc network, it computes a one-way key chain and a TESLA key chain. Then it signs the commitments of these key chains, and broadcasts them to its neighbors. A should broadcast a JOIN message to its neighbors (Figure 9.3) [14].

$$A \rightarrow *: Cert_A, Sign_A \{A \mid K^T_A(0) \mid K^F_A(0) \mid T^T_A(0) \mid T^F_A(0)\} \quad \textbf{(8.4)}$$

T^T_A (0) and T^F_A (0) are the starting times for its TESLA and TRAFFIC key chains respectively. Within the bootstrapping process every neighbor of A verifies the authenticity of A's certificate using the CA's public key, then uses node A's public key in the certificate to verify the signature on the message. If all the verifications are successful node A's key chains and their starting times are stored.

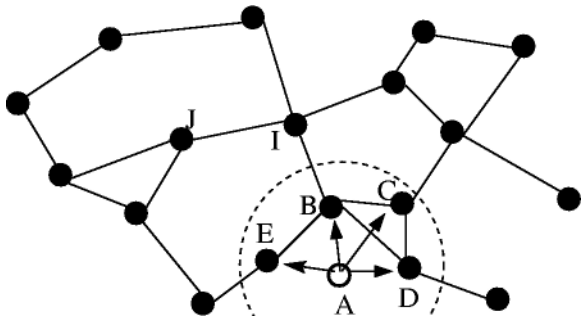


Figure 9.3: A scenario where node A joins the ad hoc network [14].

To bootstrap an authentic TRAFFIC key and a TESLA key to node A, each of its neighbors send the following ACK message to A:

$$Cert_B, Sign_B \{B \mid K^T_B(0) \mid K^F_B(0) \mid T^T_B(0) \mid T^F_B(0)\}, MAC(K^T_B(i), K^F_B(j)) \quad \textbf{(8.5)}$$

$K^F_B(j)$ is node B's most recently released TRAFFIC key, and $K^T_B(i)$ is node B's next TESLA key to be released. After receiving this message, node A first checks $Cert_B$ and $Sign_B$ to get B's authentic key chain commitments. Node A cannot verify the MAC until node B releases $K^T_B(i)$. After node A receives and verifies $K^T_B(i)$, it updates $T^T_B(0)$ and $T^F_B(0)$ to $K^T_B(i)$ and $K^F_B(j)$ and starts to communicate with node B.

9.4.4.3 Trust Maintenance

Each node in the network broadcasts KEYUPDATE messages to its neighbors to disclose the most recent TRAFFIC key. The KEYUPDATE message is authenticated with the next TESLA key in its key chain. As an example, the KEYUPDATE message node A sends can be seen in equation 8.8.

$$A, K^T_A(i-1), MAC(K^T_A(i), K^F_A(j)) \quad \textbf{(8.6)}$$

$K^F_A(j)$ is node A's most recently released TRAFFIC key, and $K^T_A(i)$ is node A's next TESLA key to be released. Node A also includes $K^T_A(i-1)$ to allow its neighbors to verify the previous KEYUPDATE messages from node A. The purpose of KEYUPDATE messages is for preventing malicious nodes from forging traffic using the TRAFFIC keys node A has already released.

It is stated that a node can disable any traffic from node A until it receives the TESLA key for authenticating the MAC or forward the traffic packets immediately which contain valid TRAFFIC keys even though it has not yet verified the MAC when it receives the KEYUPDATE message [14].

9.4.4.4 Trust Termination

There are two proposed scenarios to terminate the trust relationship between nodes. First, when a compromised node is detected, all the nodes will terminate their trust relationship with that node permanently. Second, when a node does not receive a KEYUPDATE message from a neighbor within a TESLA interval it will terminate its trust of this neighbor temporarily.

9.4.5 Security Analysis

Suppose that node E received node A's JOIN message when node A joined the network and obtained the published keys of node A's key chains. If node E has moved out of node A's transmission range for a time period of many TESLA

intervals, an outside attacker P2 can eavesdrop A's disclosed keys and use them to impersonate node A.

However, node E has not heard from node A for many TESLA intervals, it will not forward any traffic from node P2 until it receives a valid KEYUPDATE message (Figure 9.4) [14].

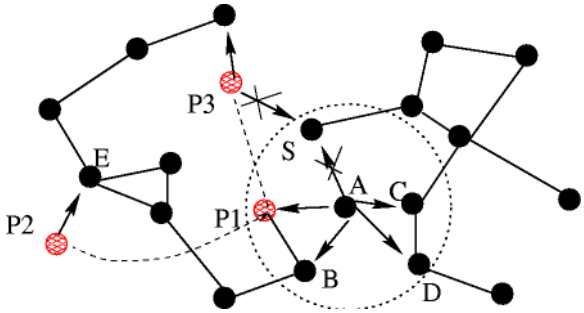


Figure 9.4: Various attacks on LHAP. PI, P2 and P3 are malicious nodes, and the dashed lines between them are private channels [14].

If there are multiple outside attackers P1 and P2 which have a private channel that allows them to communicate directly and if P1 forwards every message it gets from node A to P2 through the wormhole, P2 can rebroadcast the KEYUPDATE messages and modify the traffic packets to fake node E and node E may accept the replayed KEYUPDATE messages, and will forward the modified traffic packets for node P2.

It is stated that this attack can be detected with Global Positioning Systems (GPS) by putting node's GPS coordinates in its KEYUPDATE messages to determine if they should hear each other.

Most of the network access protocols require each node to listen to the channel before it transmitting a packet to prevent other nodes within its transmission range from transmitting simultaneously. "*When two nodes that can not hear each other send to a common neighbor at the same time, the common node will drop both packets on detection of collision. This is called hidden-terminal problem in wireless networking which can be solved by using ACKs and optional RTS/CTS control packets*" [14].

When node A is broadcasting a traffic packet that encloses a TRAFFIC key and also a malicious node P3 is transmitting a packet to node S at the same time, it is possible that node S drops both packets. At the same time node P1 can send published TRAFFIC key to node P3 through a private channel. Now node P3 can send an erroneous packet to node S impersonating node A using $K^{F}_{A}(j)$ before node A's

retransmission. However, the retransmission interval is said to be usually very small and the attackers running continuous attacks to prevent node A's retransmission can be easily detected because of the high frequency of retransmission. For a compromised node attacking the network with many traffic packets, there should be an upper bound on traffic rate (Single Insider Attack). When a compromised node shares its private key with its outside conspirators, these nodes can launch collaborative attacks without being detected. Also the compromised insiders, each of which holding a legitimate certificate can make an attack by making coalition, which could cause very damage [14].

9.4.6 Performance Analysis

There exist two types of computational overhead, which are in traffic authentication and in trust management. To verify a traffic packet there is a small computation overhead which is because of some hash functions. However, the computational overhead for trust management includes a digital signature for bootstrapping its key chains, two signature verifications for every JOIN message and one or several hash computations for every KEYUPDATE message. It can easily be seen that the computational overhead is mainly affected by the number of signature verifications.

A node adds a traffic key to every traffic packet it sends, sends a JOIN message at the time it joins the network, sends an ACK packet to every new neighbor and periodically sends a KEYUPDATE message.

"*10 bytes for a key (including a 2 bytes key id),10 bytes for a MAC, 500 bytes on average for a traffic packet, 256 bytes for a public key certificate, 128 bytes (1024 bits RSA) for a digital signature. A node joins the network for one hour, during which it sends (or forwards) 1000 packets and encounters 100 nodes. If the TESLA interval is 1 second, the traffic byte overhead is 44 bytes/s. If the TESLA interval is 2 seconds the traffic byte overhead decreases to 29 bytes/s*" [14].

LHAP is said to be independent of the network layer protocols. For some of the routing protocols which require nodes to periodically exchange routing information with their neighbors can cause LHAP to be used more efficiently by sending the KEYUPDATE messages in these messages. It avoids transmitting separate packets and causes the bandwidth and the energy for transmission and receiving not to be reduced.

As the TESLA keys are disclosed periodically in LHAP, very long key chains are required. Because for instance a network with a lifetime of five hours and TESLA

interval of one second, requires a TESLA key chain of a length of 5 * 3600 = 18,000 keys. If a node stores all the keys in a key chain, it can disclose any keys immediately, but this usage is not scalable because of its memory requirements. If a node only keeps the last key of the key chain, and every time computes the needed key from the last key in the key, it really needs high computational power. Therefore, the size of a key chain impacts both memory requirements and computation costs [14].

9.5 LEAP Efficient Security Mechanisms for Distributed Sensor Networks

9.5.1 Introduction

The proposed protocol, LEAP (Localized Encryption and Authentication Protocol) [15] is said to be a key management protocol for sensor networks that is designed to support in-network processing. It provides security properties similar to those provided by key sharing approaches.

LEAP includes support for multiple keying mechanisms where different types of messages exchanged between sensor nodes have different security requirements. The proposed protocol supports four types of keys for each sensor node which are an individual key shared with the base station, a key shared with another sensor node, a cluster key shared with multiple neighboring nodes, and a group key shared by all the nodes in the network. LEAP also includes an efficient protocol for inter-node traffic authentication based on the use of one-way key chains. A proposed feature of the authentication protocol is that it supports source authentication without preventing passive participation.

For this framework it is assumed that the sensor network is static and not mobile. There exists a base station which acts as a controller (or key server) which is assumed to be a device with high computational capabilities and long lasting power. The sensor nodes are supposed to be similar in computational and communication capabilities and power resources which are like the Berkeley MICA motes.

The main security requirements of LEAP not only include authentication and confidentiality but also robustness and survivability. It should be possible to add new sensor nodes incrementally to the sensor network.

9.5.2 Leap: Localized Encryption and Authentication Protocol

As it is stated that the packets exchanged by nodes in a sensor network can be classified into several categories like control packets versus data packets, broadcast packets versus unicast packets, etc., the security requirements for a packet will depend on the category it falls in. Authentication is required for all type of packets but confidentiality may only be required for some types of packets.

The algorithm argues that single keying mechanism is not appropriate for all the communication that is needed in sensor networks.

9.5.2.1 Individual Key

Every node has a unique key that it shares with the base station. This key is used for secure communication between a node and the base station. For example, when a node wants to send an alert to the base station for an unexpected behavior, it uses this key. And also the base station can use this key to encrypt any sensitive information, like keying material.

9.5.2.2 Group Key

This is a globally shared key that is used by the base station for encrypting messages that are broadcast to the whole group.

9.5.2.3 Cluster Key

A cluster key is a shared key of a node and all its neighbors which is mainly used for securing locally broadcast messages. In LEAP each node owns a unique cluster key that it uses for securing its messages, while its immediate neighbors use the same key for decryption or authentication of its messages.

9.5.2.4 Pair Wise Shared Key

Every node shares a pair wise key with each of its immediate neighbors. In LEAP, pair-wise keys are used for securing communications that require privacy or source authentication.

9.5.2.5 Key Establishment

• N is the number of nodes in the network.

• u, v are principals such as communicating nodes.

• $\{f_k\}$ is a family of pseudo-random functions.

• $\{s\}_k$ means encrypting message s with key k.

• MAC(k, s) is the message authentication code (MAC) of message s using a symmetric key k.

From a key K a node can derive other keys for various security purposes. For example, a node can use $K0=f_k(0)$ for encryption and use $K1 = f_k(1)$ for authentication. The message is encrypted with K0 and authenticated with K1 respectively [15].

9.5.2.6 Establishing Individual Node Keys

Every node has a special key which is only shared with the base station which is generated and loaded into each node before its deployment.

"*The individual key* K^m_u *for a node u is generated using* $K^m_u = f_k{}^m_s$ *(u). Here f is a pseudo-random function and* k^m_s *is a master key known only to the controller. The controller might only keep its master key to save the storage for keeping all the individual keys. When it needs to communicate with an individual node u, it computes* K^m_u *on the fly*" [15].

9.5.2.7 Establishing Pair-wise Shared Keys

For nodes whose neighborhood relationships are predetermined, the key establishment is simply done by loading the sensor nodes with the corresponding keys before the deployment.

9.5.2.8 Key Pre-distribution

The controller generates an initial key K_I and loads each node with this key. Each sensor node u derives a master key $K_u = f_{KI}(u)$.

9.5.2.9 Neighbor Discovery

When it is deployed, node u first initializes a timer to fire after time T_{min}. It then tries to discover its neighbors. It broadcasts a HELLO message which contains a nonce, and waits for each neighbor v to respond with its identity. The reply from each neighbor v is authenticated using its master key K_v. Since node u can compute K_v using K_I, it is able to verify node v's identity independently [15].

9.5.2.10 Pair wise Key Establishment

Node u computes its pair-wise key with v, K_{uv}, as $K_{uv} = f_{kv}(u)$. Node v can also compute K_{uv} independently. No message is exchanged between u and v in this step. It is stated that node u does not have to authenticate itself to node v explicitly and immediately, because any new messages authenticated with K_{uv} by node u will prove node u's identity [15].

9.5.2.11 Inter-node Traffic Authentication

Within this framework for a secure sensor network every message should be authenticated before it is forwarded or processed and the authentication scheme should be lightweight for a sensor node. The proposed algorithm tried to use μTESLA [16] for broadcast authentication for the controller which needs loose time synchronization. μTESLA builds the authentic message by using one-way key chain and delayed key disclosure. It is stated that μTESLA is not suitable for inter-node traffic authentication because of the delayed key disclosure which makes immediate authentication impossible.

In μTESLA the receiver has to wait for one μTESLA interval to get the MAC key of the previous interval, so by using this protocol a message traversing n hops will reach its destination at least after n μTESLA intervals. Another disadvantage is that a sensor node has to buffer all the unverified packets. This protocol can not be used for authenticating all traffic in the network because of this latency and buffering storage requirements.

The first proposed solution for these drawbacks is using pair wise keys to provide source authentication and using cluster keys. Every node authenticates a packet it sends using its own cluster key as the MAC key and a receiving node verifies the packet using the same cluster key it obtained from the sending node in the cluster key establishment phase. Then it authenticates its packet to its own neighbors with its own cluster key and this causes a message gets authenticated repeatedly in hop by hop basis. In this approach it can be seen that the communication overhead is really small where a node only adds a MAC to each packet. However, within this approach insider attacks are possible after the adversary compromises a sensor node [15].

9.5.2.12 One-way Key Chain Based Authentication

As it is known from the previous proposed protocols every node generates a one-way key chain of a certain length and transmits the first key of the key chain to each neighbor, encrypted with their pair wise shared key. Whenever a node has a message

to send, it attaches the next key in the node's one-way key chain which is called the AUTH key. When a neighbor receives the message it can verify its authenticity based on the recently disclosed AUTH key of the sending node, because the AUTH keys are disclosed in an order reverse to their generation.

In this framework a node only needs to authenticate a packet to its immediate neighbors, because authentication is done in every hop. For example, an adversary can jam node v by transmitting dummy packets to v at the same time when node u is transmitting. After that the adversary can send a modified packet to node v impersonating node u. As node v has not received a packet with the same AUTH key, it will accept the modified packet. "*For this attack, if it is launched by an outsider node, it can be prevented by combining the AUTH keys with the node's cluster key, and using this combined keys as MAC keys for authentication. Then the outsider node can not launch such attack as it does not know node's cluster key*" [15].

9.5.3 Performance Evaluation

9.5.3.1 Computational Cost

While updating the cluster key, the node encrypts its new cluster key using the pair wise key shared with each neighbor. Therefore, the number of encryptions is determined by the number of neighbors. If n is the number of neighbors of the node updating the cluster key, and di, i = 1,2,..., n is the number of legitimate neighbors of each of these do neighbors, then the total number of encryptions and decryptions are d1+d2…dn.

9.5.3.2 Storage Requirements

It is stated that if a node has d neighbors, it stores one individual key, d pair wise keys, d cluster keys and one group key. For the inter-node authentication, the most recent AUTH key of each neighbor and its own one-way key chain should also be stored.

9.5.4 Conclusions

LEAP (Localized Encryption and Authentication Protocol) is stated to be a key management and authentication protocol for sensor networks with the following properties:

• The proposed protocol states that the different types of messages have different security requirements.

• LEAP uses one-way key chains for inter-node traffic authentication which makes the protocol more efficient.

• The key establishment and key updating procedures are efficient and the storage requirements per node are small.

9.6 Efficient and Secure Source Authentication for Multicast

This algorithm addresses the problems in transmission of data to multiple receivers in sensor networks. In recent years several multicast routing protocols have been proposed to avoid sending the data separately to each receiver, more in the IP layer. The security of multicast communication has a lot of difficulties which are not seen in unicast communication. Tesla focuses on providing source authentication for multicast communication [13].

For unicast communication the standard point-to-point authentication mechanisms like using message authentication codes are enough, but these mechanisms do not fit well to multicast communication. The MAC which is appended to the packet provides source authentication if the receiver's clock is synchronized with the sender's time. The proposed algorithm mostly solves the problems of using TESLA.

"*In TESLA the receiver buffers packets, until the sender discloses the corresponding key before authenticating the packets which can cause storage problems and can make it open to denial-of-service (DoS) attacks. The proposed method allows receivers to authenticate most packets immediately upon arrival by using one extra hash per packet*" [13].

For different receivers which cause different delay times, TESLA uses multiple keys for authentication with different disclosure delay times. The proposed algorithm achieves the same functionality with less overhead per packet.

In TESLA the sender performs authenticated time synchronization with every receiver, which is not suitable for multicast groups. The proposed method uses only a single public-key operation per time-unit which is regardless of the number of the receivers that need synchronizing during this time unit. In the proposed algorithm there are two time synchronizations, where the receiver synchronizes its time directly

with the sender, and both the sender and receiver synchronize their time with a time synchronization server.

TESLA assumes that all members have synchronized with the sender before any transmission starts but this algorithm allows a receiver to join into an ongoing session and to synchronize at a later time.

9.6.1 Sender Setup

The sender sends message chunks $\{M_i\}$ in one network packet P_i and it splits the time into even intervals I_i where the duration of each time interval is T_{int}, and the starting time of the interval I_i is T_i, so $T_i = T_0 + i * T_{int}$ holds. In each interval, the sender may send zero or multiple packets.

Before sending the first message, the sender determines the sending duration, the interval duration, and the number of keys of the key chain which is a one-way chain. The sender selects the last key K_N of the key chain randomly and computes the entire chain using a random function F. F is a one-way function where each element of the key chain is computed as $K_i = F(K_i+1)$.

Each key can be derived from K_N as $K_i = F_{N-i}(K_N)$ where K_i is active in interval I_i.

The proposed algorithm does not use the same key multiple times in different cryptographic operations where there is a second pseudo-random function F' which helps to generate the key ($K_i' = F'(K_i)$) which is used to compute the MAC of messages in each interval. "*The algorithm uses HMAC in conjunction with a cryptographically secure hash function for the pseudo-random function where F(x) = HMAC(x,0) and F'(x) = HMAC(x,1) (0 and 1 are 8-bit integers that represent the data, x is the key)*" [13].

9.6.2 Bootstrapping a New Receiver

In TESLA to accept a new receiver an initially authenticated data packet is required. This authentication is achieved with a digital signature scheme like RSA or DSA.

The initial authenticated packet contains the following information for synchronizing the time direct and indirect.

• The beginning time of a specific interval T_j, along with its id I_j

• The interval duration T_{int}

• Key disclosure delay d (unit is interval)

• A commitment to the key chain K_i (i < j - d where j is the current interval index)

9.6.3 Sending Authenticated Packets

For computing the MAC of the messages that are sent in the same interval the same key is used because each key of the key chain is used in one time interval. The key remains secret for d-1 future intervals. In interval I_j key K_{j-d} is disclosed and when the receivers receive that key, they can verify the authenticity of the packets that are sent in interval I_{j-d}. The construction of packet P_j sent in interval I_i is: { M_j | MAC(K_i',M_j) | K_{i-d} } [13].

9.6.4 Receiver Tasks

The receiver should be sure that the sender is not in the corresponding key disclosure time interval. If a packet satisfies this security condition, then it is certain that the message is not altered in transit, because the corresponding MAC key is not yet disclosed. Otherwise the receiver must drop that packet, because the authenticity does not hold.

When the receiver receives packet P_j sent in interval I_i at local time t_c it computes an upper bound on the sender's clock t_j to verify that the sender is not in the disclosure interval of the key K_i. The receiver then stores the packet to verify the authenticity later when it knows K_i. The unauthenticated message M_j can be sent to the application, and notified through a callback when M_j is verified. Otherwise M_j is buffered until it is authenticated and then sent to the application.

"*If the packet contains a disclosed key K_{i-d}, the receiver uses it to authenticate previous packets. If the last key value in the reconstructed key chain is K_v. The receiver verifies if K_{i-d} is legitimate by verifying that. $K_v = F^{i-d-v}(K_{i-d})$. If that condition is correct, the receiver updates the key chain which allows it to verify the previously received packets*" [13].

9.6.5 Extensions

9.6.5.1 Immediate Authentication

In TESLA the receiver needs to buffer packets during one disclosure delay before it can authenticate them and this is not practical because of the need for buffer space which might cause packets to be dropped. The proposed algorithm solves buffering problem by using an immediate authentication method.

The sender buffers packets during one disclosure delay and stores the hash value of the data of a later packet in an earlier packet. And when the receiver authenticates the earlier packet the later packet is also authenticated by the hash value.

If the sender sends out a constant number v of packets per time interval and the sender appends the hash value of the message chunk M_{j+vd} to M_j in time interval T_j, and then computes the MAC value also over $H(M_{j+vd})$ with the key K_i.

When the packet P_{j+vd} arrives at the receiver with the disclosed key K_i it authenticates packet P_j sent in interval I_i. P_j includes hash of the data M_{j+vd} in P_{j+vd} and if P_j is authentic, $H(M_{j+vd})$ is also authentic and the data M_{j+vd} is immediately authenticated. If P_j is lost P_{j+vd} can still be authenticated later using the MAC value (Figure 9.5) [13].

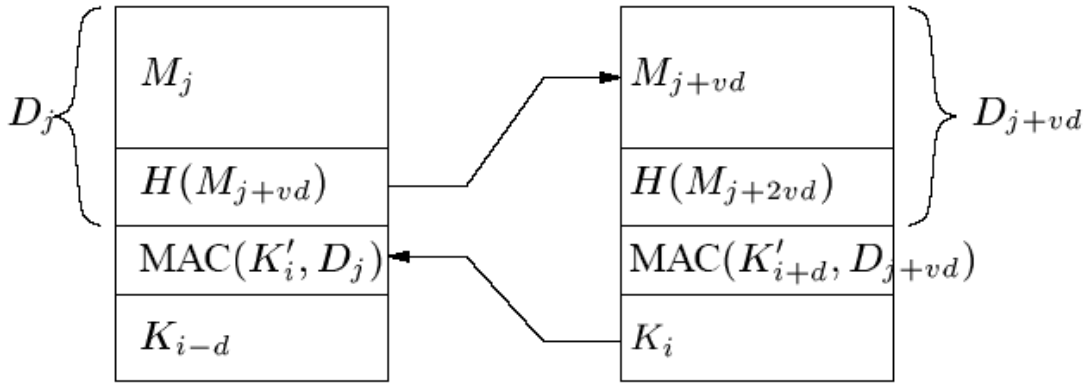


Figure 9.5: Immediate authentication packet example [13].

9.6.5.2 Concurrent TESLA instances

Receivers with a low network delay and short key disclosure delay causes short authentication delay, but receivers with a long network delay could not work this time. There is a need to use multiple instances of TESLA with different disclosure delays where each receiver can decide which disclosure delay or TESLA instance to

use. If there is one key chain per instance, and each instance requires 20 bytes per packet (80 bit for key disclosure and 80 bit for the MAC value), using three instances causes 60 bytes overhead per packet.

The defined algorithm proposes to use the same key chain with a different key schedule for all instances. All instances share the same time interval duration and the same key chain.

9.6.6 Time Synchronization Issues

Loose time synchronization requirement secures TESLA against an active attacker that allows the receiver to compute an upper bound of the difference between the sender's local time and the receiver's local time.

9.6.6.1 Direct Time Synchronization

In direct time synchronization, the receiver synchronizes its time with the sender. A simple two-phase protocol that satisfies the requirements is proposed for the algorithm.

In the protocol, the receiver first records its local sending time t_R and sends a time synchronization request with nonce to the sender. The sender records its local receiving time t_S and sends a signed response packet containing t_S and the nonce to the receiver after receiving the time synchronization request (Figure 9.6).

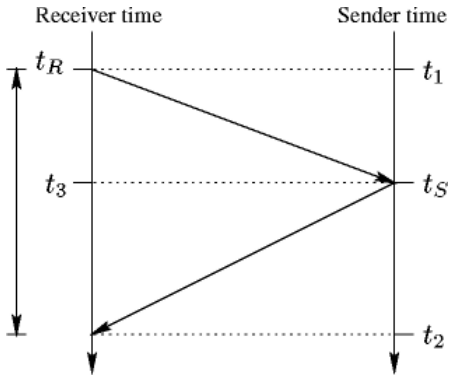


Figure 9.6: The receiver synchronizes its time with the sender [13].

9.6.6.2 Indirect Time Synchronization

In indirect time synchronization, both the sender and the receivers synchronize their time with a time reference. "*If the receiver is already time synchronized with the time reference, the sender periodically broadcasts digitally signed packets that contain its*

time synchronization with the time reference, the time interval and key chain information, along with the sender's maximum synchronization error. A new receiver starts authenticating the data after it receives one of the signed packets" [13].

9.6.6.3 Delayed Time Synchronization

Another time synchronization requirement is that, the receiver can receive the data from the sender and after some time, it synchronizes its time and authenticates the data. The receiver only stores the arrival time of each packet, and it authenticates the packet after it performed the time synchronization.

9.6.7 Security Discussion and Robustness to DoS

A DoS attack on the sender is not possible if TESLA is used with indirect time synchronization, because the sender does not perform per-receiver operations. A DoS attack is possible in direct time synchronization, since an attacker can flood the sender with requests.

Some DoS attacks on the client such as delay or drop packets are possible. Delay packets could cause packets not to be authenticated. The duplicated packet is only accepted by the receiver if it comes with a short delay. To prevent this attack sequence numbers can be added to each packet in the MAC. According to the proposed protocol duplicate packets will be filtered in the network layer or in the application layer.

There can also be bogus traffic to flood the receivers. If the receivers have certain size buffers, they have to buffer packets for one disclosure delay time until the packets can be authenticated. Thus the buffer size should be the multiplication of the network bandwidth and the disclosure delay time. For 10Mbps network connection with a 500ms disclosure delay 640kB buffer size is needed. If the receiver's buffer size is not large enough flooding could result in dropping packets. The algorithm suggests using a shared secret key in all receivers and adding a MAC to each packet with the shared secret key which enables verifying the packet instantly [13].

If an attacker sends a packet that is far in the future, the receiver may try to verify the key disclosed in the packet by using the pseudo-random function to reach the last known key chain value. However this kind of attack is not important because the receiver can easily check that the packet interval is less or equal to the latest interval. The incoming packet can be dropped if its sending interval I_j is not in the future.

9.6.8 Conclusions

The proposed algorithm is an extension to TESLA which provides source authentication in a loosely time synchronized environment. The proposed protocol has low computation and communication overhead with providing immediate authentication and prevents some denial-of-service attacks.

9.7 A Novel Authentication Scheme for Ad hoc Networks

The proposed authentication algorithm is for a hierarchical architecture where a cluster based network has been used.

9.7.1 Overview of the Clustering Phenomenon

"*The cluster based architecture minimizes the flooding of route discovery packets which uses Cluster Based Routing Protocol (CBRP) and is suitable for large networks with several nodes. The network is divided into a number of overlapping or disjoint 2-hop-diameter clusters and a cluster head is selected for each cluster to maintain the membership information*" (Figure 9.7) [17].

A cluster can be identified by its cluster Head ID where each node knows its Cluster Head(s). A node is said to be in cluster A if it has bi-directional link to the cluster head and according to CBRP, the node with lower node ID is the cluster head.

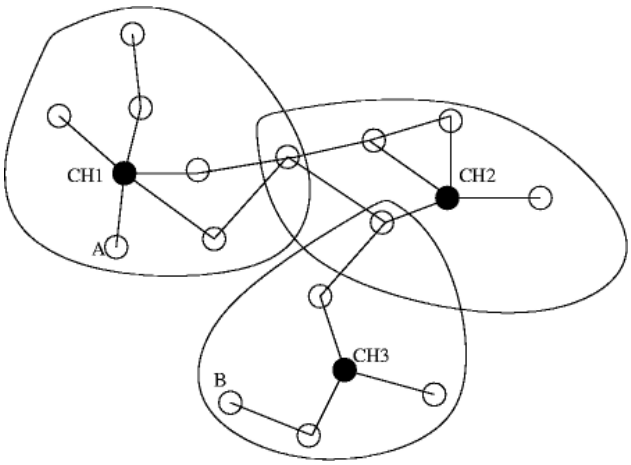


Figure 9.7: Example of network divided into clusters [17].

All nodes in the network broadcast HELLO messages periodically which contain information about the neighboring nodes and adjacent clusters. They also help the nodes to know the topology [17].

9.7.2 Assumptions

All the nodes mutually trust each other, have required computational power to execute the encryption and key generation algorithms, have enough memory to store the keys and use TCP.

9.7.3 Key Definitions and Distribution Methodology

For a newly joining node, it is given a system public key and system private key which are shared by all the nodes of the network. Each node also needs a cluster key which is said to be unique to each cluster and shared by all the nodes in the cluster. The cluster key which is encrypted with the system public key is generated by the cluster head and distributed to all the cluster members.

Each cluster head also has a unique public and private key pair called head key where the private key is known only by the generating node and the public key is known by all the nodes. When a node is elected as the head, the public key that it generates is broadcasted by a network wide broadcast. So each node stores a pair of system keys, a cluster key and a table consisting of cluster ids and the corresponding head's public key. The cluster head also stores its private key [17].

9.7.4 Proposed Steps

• When a node joins a network for the first time: The system key pair is used for authentication between the joining node and an existing member of the network. When a new node joins the network it sends hello messages, gets detected by the cluster head and finally it retrieves the cluster key and the table containing the cluster ids and head public keys.

• When a node leaves a cluster and joins another cluster: If a node moves from a cluster to new one, it is treated as a new node joining a cluster. The old cluster purges the entry for this node when it doesn't receive hello message for a certain predefined time interval.

• When a node from a cluster wishes to communicate with a node belonging to another cluster: For complete confidentiality of the message, the entire packet has to be encrypted with a session key which is shared by the two parties involved in the communication. But sometimes where confidentiality is not important then it is not necessary to encrypt the whole packet. In that case for achieving authentication, appending a small encrypted tag to each packet is enough.

9.7.5 Description of the Algorithm

The cluster head is the certification authority for all its members. If A and B want to communicate with each other, they exchange a session key that is only valid for one session where the head's keys are used for secretly exchanging session keys. The Cluster Heads then decrypt and transmit the session key to their corresponding members who are involved in the session.

"*If a node wants to establish a session with another node, it also sends this request to the head and the head generates a set of k random prime numbers where k can be 16 or 32. The k numbers (R_1, R_2... R_k) are encrypted first with the head's private key and then with the cluster key*" [17]. It is stated that along with each number a time-stamp is encrypted so that they could be used for a limited amount of time. Therefore, each cluster head has a table containing $E_ck(E_pv(R_1,t_v))$ to $E_ck(E_pv(R_k,t_v))$ where E_ck is encryption using cluster key, E_pv is encryption using the head's private key and t_v is the corresponding time-stamp [17].

Then the k encrypted values are broadcasted by the head where all other cluster members buffer these k values which serve as authentication tags for any of the members. The tags are decrypted with the cluster key before they are buffered. They can be used as authentication tags because they have been encrypted with the head's private key. They are also encrypted with the session key to protect them from malicious listeners.

It is stated that when a window of w packets is to be sent, the k encrypted tags are used to obtain a permutation of size w. Each of these w tags is appended to one packet. When the receiver gets the packets with tags appended, it should be able to verify the origin and authenticity of the tags by using a check function where tags are input to the function, and a value that is unique for each set of input is output from the function. The function can be as simple as the product of the decrypted tags because the tags are prime numbers.

The sender uses the check function like check(R_0,R_2..$R_{(m-l)}$) for m inputs at a time and the output of the function is encrypted. The highest sequence number among these 'm' packets is also encrypted along with the value obtained from the check function. The session key is used for this encryption. When the receiver receives the packets, it also computes the check function of the received tags. The computed value is compared with that sent by the sender. If they match the sender accepts the packet.

As the check function is computed for every 'm' packets, the receiver could even narrow down the search for unauthentic packets to a range of 'm'. In the proposed algorithm the checksum field of the header is also encrypted with session key so that any errors in the data during transit can be detected by computing checksum [17].

9.7.6 Advantages and Limitations of the Proposed System

The tags are encrypted with the head's private key and by using the check function the source of these tags is verified. It is stated that if the same pattern of tags and check function are reused, the sequence numbers do not match. If the sequence numbers are replayed, they would be rejected as duplicates.

The encryption of the checksum field helps in ensuring data integrity. Cluster head needs to compute the tags once and could use it for all its cluster members which reduce the number of encryptions. The tags are also broadcast and therefore transmission delay is also reduced. Each node needs to maintain the keys of just the cluster heads with less memory usage.

The cluster head needs to generate random prime numbers periodically. The permutations of tags need to be obtained so that a tag could be appended to each packet. Each node should be capable of running an algorithm that generates a random pattern of a specified length. A session key should be generated for each session. The cluster head needs to be a powerful node in order to perform all its functions.

9.8 On Communication Security in Wireless Ad-Hoc Sensor Networks

It is known that generally three types of data sent through the network which are, sending mobile code, sending locations of sensor nodes and sending application specific data.

"*According to these types there exist some threats. Malicious mobile code injected into a network can change the behavior of the network. Acquiring locations of sensor nodes can make an adversary to know locations of sensor nodes. Protection of application specific data depends on the security requirements of a particular application*" [18]. The main goal of this framework is to minimize security related energy consumption.

9.8.1 Communication Security Scheme

Implementing security mechanisms in a sensor network creates additional overhead with latency increases and energy consumption increases. To minimize the security related costs sensitivity of the encrypted information is important. For the types of data in the network, three security levels are defined.

"*Security level I is for mobile code which the most sensitive information and security level II is dedicated to the location information in messages, the security level III is applied to the application specific information. The strength of the encryption for each of security levels corresponds to the sensitivity of the encrypted information where the encryption at level I is stronger than the encryption at level II and so on*" [18].

In this framework different security levels are implemented by using different algorithms or the same algorithm with some adjusted parameters. Using one algorithm with adjustable parameters has the advantage of occupying less memory and program space and RC6 is selected for the proposed framework. RC6 has an adjustable parameter (number of rounds) that affects its strength and if the strength increases then the overhead also increases.

The SensorWare architecture suggests using group keys. Otherwise, if each pair of nodes uses a pair of keys, communication becomes unicast based the number of messages increases. All nodes in the network share an initial number of master keys where the number of keys depends on the estimated lifetime of the network. It requires that the protocol guarantees that all nodes received a key. One of the keys from the list of master keys is active at any moment. According to the algorithm there is a pseudorandom generator running at each node for the selection of a particular key. Periodically and synchronously each node generates a new random number which is used to provide index to an entry in the table of the available master keys. This entry contains the active master key. The keys for three levels of security corresponding to the three types of data are then derived from the active master key [18].

9.8.1.1 Security Level I

As the messages that contain mobile code are less used, strong encryption can be used in spite of the high overhead. Nodes use the current master key for protection at this security level. The set of master keys and the pseudo-random number generator are needed in order to insert any code into the network.

9.8.1.2 Security Level II

A novel security mechanism that isolates parts of the network is proposed for the data that contains locations of sensor nodes. In the proposed framework the overhead of the encryption of the location information affects the overall security overhead in the network because the locations of sensor nodes are included in most of the messages. The protection level is lower for the location information than for mobile code and the key for the level II is weaker. For making the system more secure, location-based keys are used for level II encryption which enables separation between some regions.

"*The area covered by a sensor network is divided into cells where nodes share a common location-based key, which is a function of a fixed location in the cell and the current master key. There is also a special area between the adjacent regions whose width is equal to the transmission range. Nodes in these regions stores the keys for all adjacent cells which ensure that two nodes within a transmission range from each other have a common key*" [18].

If the area is divided in uniformly sized cells, a node can determine its cell membership fastly and easily. Thus the network is divided into hexagonal cells where the gateway nodes have at most three keys. An extended cell is a hexagonal cell, which has the same center as the original cell and the distance between its sides and the sides of the original cell is equal to the transmission range of the sensor nodes. If a node is within the extended cell of C_x, it will have the key of C_x, K_{Cx} (Figure 9.8).

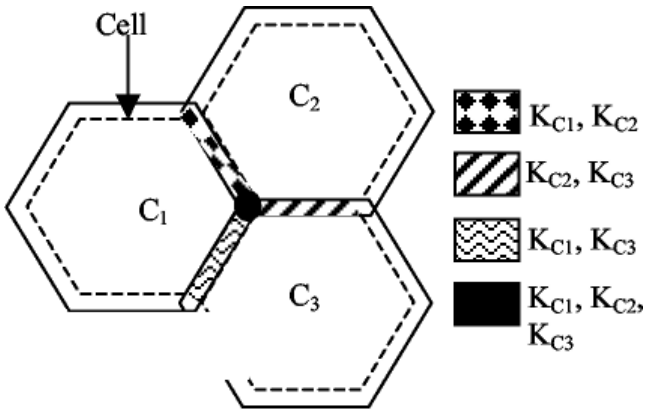


Figure 9.8: Cells, Extended cells and areas with multiple keys [18].

9.8.1.3 Security Level III

For the application specific data, a weaker encryption than the others which requires lower computational overhead is used. As there are lots of messages with application specific data transferred within the network, an algorithm with less computational overhead is used by decreasing the strength of security. The key for the encryption of the level III is derived from the MD5 hash function which accepts the master key as input. The master key is periodically changed and the corresponding key at this level also changes. It is assumed that the sensor nodes are time synchronized and have knowledge of their location [18].

9.8.2 Implementation

The encryption routines of RC6 are ported on the Rockwell WINS sensor nodes and by using the ARM System Developers Kit profiling tools, the clock cycles spent for encryption and decryption of a single 128 bit block with a key of length 128 are calculated, versus the number of rounds. Figure 9.9 depicts the total clock cycles for encryption and decryption of a single 128-bit block with a 128-bit key versus the number of rounds.

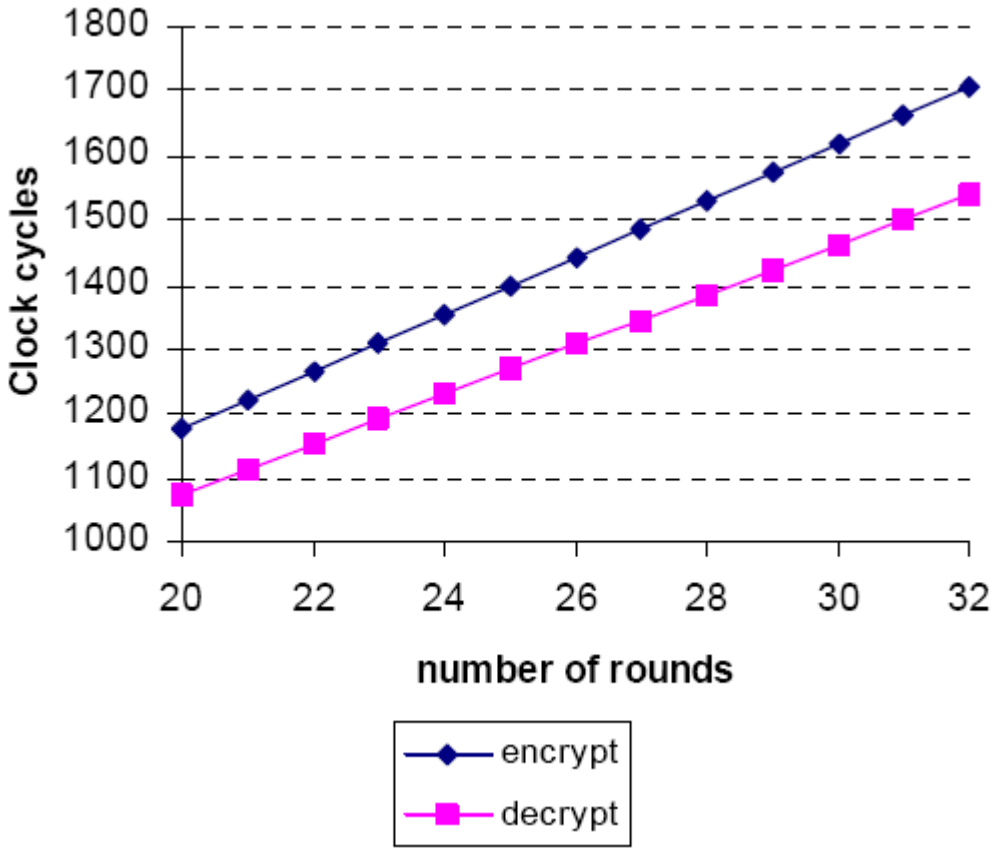


Figure 9.9: Encryption and decryption clock cycles versus the number of rounds for RC6 [18].

9.8.3 Conclusion

The protocol ensures authentication of broadcast messages by distributing a key after the messages encrypted with that key. In the proposed architecture all nodes can be senders and receivers of broadcast messages and to achieve strong authentication offered by μTESLA, each node should have its own key known to all other nodes in the network. This solution does not scale well in a network with thousands of nodes.

If the mechanism is chosen according to the most sensitive data in the network, security related resource consumption might be unacceptable. But a less consuming mechanism could be acceptable for some serious security threats.

9.9 SPINS: Security Protocols for Sensor Networks

SPINS consists of two secure building blocks called SNEP and μTESLA. SNEP provides data confidentiality, two-party data authentication, and data freshness. As it is known that providing efficient broadcast authentication for sensor networks is a hard problem, a new protocol which is called μTESLA is proposed for resource constrained environments [16].

The recent secure algorithms, which were designed for powerful workstations can not be used within resource constrained sensor network systems. For example, the working memory of a sensor node is insufficient to even hold the variables that are used in asymmetric cryptographic algorithms. The new protocol which is called μTESLA is developed by extending and adopting TESLA for making it practical for broadcast authentication in sensor networks.

"*Wireless systems are generally un-trusted because someone can eavesdrop on the traffic, inject new messages or replay and change old messages. As the base stations are necessary parts of the trusted computing base, they are the useful ways for the nodes to communicate with another center. Each trusted node is given a master key which is shared with the base station and each node trusts itself and its sensors. Also the local clock should be accurate*" [16].

X and Y are communicating nodes, N_A is a nonce generated by X, $M_1 \mid M_2$ denotes the concatenation of messages M_1 and M_2 are messages and K_{XY} denotes the secret (symmetric) key which is shared between X and Y.

$\{M\}\ K_{XY}$ is the encryption of message M with the symmetric key of X and Y.

{M} (K_{XY}, IV) denotes the encryption of message M, with key K_{XY}, and the initialization vector IV which is used in encryption modes such as cipher block chaining (CBC), output feedback mode (OFB), or counter mode (CTR).

9.9.1 SNEP

SNEP has low communication overhead that only adds 8 bytes per message and it uses a counter. SNEP provides semantic security which prevents attackers from getting the original message content from the encrypted message. SNEP also provides data authentication, replay protection, and message freshness [16].

According to the proposed algorithm semantic security is provided by encrypting each different message depending on a counter value which is different each time and long enough not to repeat within the lifetime of the node.

Data authentication is achieved by using MACs. Replay attacks are prevented by the help of the counter values. The counter state is kept at each end point and does not need to be sent in each message which helps to decrease the communication overhead.

Normally SNEP only provides weak data freshness, because it only enforces a sending order on the messages within node B, but there is no guarantee that the message received by node A is the message that was created by B in response to an event in node A.

Node A achieves strong data freshness for a response from node B by using a nonce N_A (which is a random number sufficiently long such that it is unpredictable). Node A generates and sends N_A with a request message R_A to node B. And node B returns the nonce with the response message R_B in an authenticated protocol to achieve strong data freshness. The proposed protocol uses the nonce implicitly in the MAC computation instead of returning the nonce to the sender and if the MAC is verified correctly, node A knows that node B generated the response after it sent the request [16].

9.9.2 µTESLA: Authenticated Broadcast

The recently proposed TESLA protocol provides efficient authenticated broadcast but is not designed for such limited computing environments. The new designed µTESLA solves some of the inadequacies of TESLA. TESLA authenticates the initial packet with a digital signature, which is too expensive for the sensor nodes.

On the other side µTESLA uses symmetric mechanisms. It is expensive to store a one way key chain in a sensor node. µTESLA restricts the number of authenticated senders.

9.9.3 µTESLA Overview

µTESLA requires that the base station and the nodes are loosely time synchronized, and each node needs to know an upper bound on the maximum synchronization error. To send an authenticated packet, the base station computes a MAC on the packet with a key that is secret at that time and when the node gets the packet, it should verify that the corresponding MAC key was not yet disclosed (using maximum time synchronization error). As the MAC key is only known by the base station and it is certain that no adversary could have changed the packet in transit, the node stores the packet in a buffer. When the base station broadcasts the verification key to all receivers and when the node receives the disclosed key, it can easily verify the authenticity of the key and use this key to authenticate the packet stored in its buffer. In the most of the algorithms a key of a key chain is generated by a publicly known one-way function F, the sender chooses the last key K_n of the key chain and continuously applies F to compute all other keys using $K_i = F(K_{i+1})$ (Figure 9.10) [16].

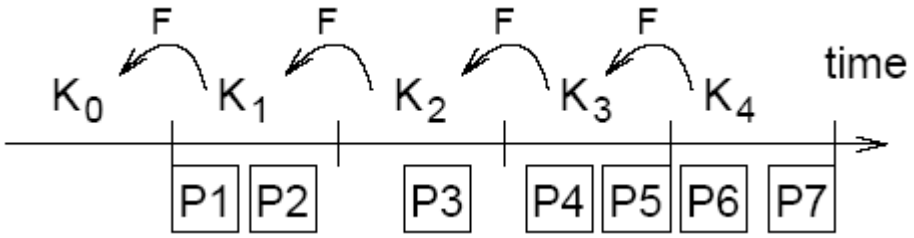


Figure 9.10: Using a Time-Released Key Chain for Source Authentication.(K keys, P packets) [16].

The key disclosure is independent from the packets, and is tied to time intervals instead of appending the disclosed key to each data packet which makes µTESLA more effective. According to µTESLA, the sender broadcasts the current key periodically in a special packet.

For sender setup, the sender generates the one-way key chain of length n by selecting the last key K_n randomly, and generating other keys by applying the one-way function F where F can be a cryptographic hash function such as MD5.

It is stated that when the receiver has an authenticated key of the chain it can compute other keys of the chain easily which are self authenticating. For example, if

the receiver has an authenticated value K_i of the key chain, the receiver can easily authenticate K_{i+1}, by verifying $K_i = F(K_{i+1})$.

Another requirement of μTESLA is that the sender and receiver are loosely time synchronized. According to the proposed algorithm, the time is divided into small intervals where one of the keys of the key chain is used in one time interval and in interval t the sender computes the message authentication code (MAC) of packets by using the current key K_t. After some time the key K_t is disclosed and the packet can be verified [16].

The receiver should be sure that the key is not yet disclosed and no adversary could have forged the contents which is the security condition of the proposed algorithm. Therefore the sender and the receiver need to be loosely time synchronized and the receiver needs to know the key disclosure schedule. If the incoming packet satisfies this security condition the packet is stored by the receiver to be verified after required key is disclosed otherwise the packet is dropped, because the packet might have been altered.

It is also not possible to store the keys of a one way key chain as the nodes are memory limited and recomputing each key from the initial key is computationally expensive. To deal with this problem the node broadcasts the data through the base station by using SNEP. The other solution to the problem can be storing the one way key chain by the base station and sending them to the broadcasting node as needed. Maybe the base station can also broadcast the disclosed keys.

9.9.4 Implementation

It is stated that RC5 from OpenSSL is used after tuning the code and an additional 40% reduction in code size is achieved. The same function is used both for encryption and decryption to save code space where the counter (CTR) mode of block ciphers has this property.

CTR mode requires a counter for proper operation where this counter value should not be reused for not to degrade security. "*By using this approach when the sender and receiver share the same counter value, the counter from the message can be omitted. When the two nodes lose the counter synchronization, they can transmit the counter to resynchronize using SNEP with strong freshness*" [16].

The CTR encryption provides weak freshness by incrementing the counter value after each message transmission and the receiver verifies that received messages

have an increasing counter. For achieving strong freshness, a random nonce N_M (a 64-bit value that is unpredictable) is created and appended in the request message. Then when the receiver generates the response by including the nonce in the MAC computation, the sender verifies the MAC of the response and verifies that the response was generated after it sent out the request message.

For random number generation it is stated that using random digits from sensors or from radio receiver is not efficient as they consume a lot of power. Instead, the MAC function is used as the pseudo-random number generator (PRG), with the secret pseudo-random number generator key.

The proposed algorithm uses the CBC-MAC for message authentication. MAC is used to check both authentication and integrity of messages, so CRC like error checking mechanisms are not used. Finally, for key setup a secret master key is shared by the base station and the node [16].

9.9.5 Performance

It is stated that the used sensors support a maximum throughput of twenty 30-byte messages per second where the microcontroller is idle for 50% of the time. With this throughput and assuming a single key setup, one MAC operation, and one encryption operation the algorithm is stable and able to encrypt and sign every message (see Table 9.2).

Table 9.2: Performance of security primitives in TinyOS

Operation	No. of Instructions
MAC (16 byte message)	600
Encrypt (16 byte message)	120
Key setup	8000

"*When energy costs of adding security protocols to the sensor network is investigated, it is clear that most of the overhead is caused by the transmission of extra data rather than computational costs*" (Figure 9.11) [16].

9.9.6 Conclusion

It is seen that the communication costs are small because the authentication, freshness, and confidentiality require transmitting 8 bytes per unit. It is easy to guarantee authentication, freshness, and confidentiality on a per packet basis. It is also stated that it is difficult to improve this scheme, as transmitting a MAC is fundamental for guaranteeing authenticity.

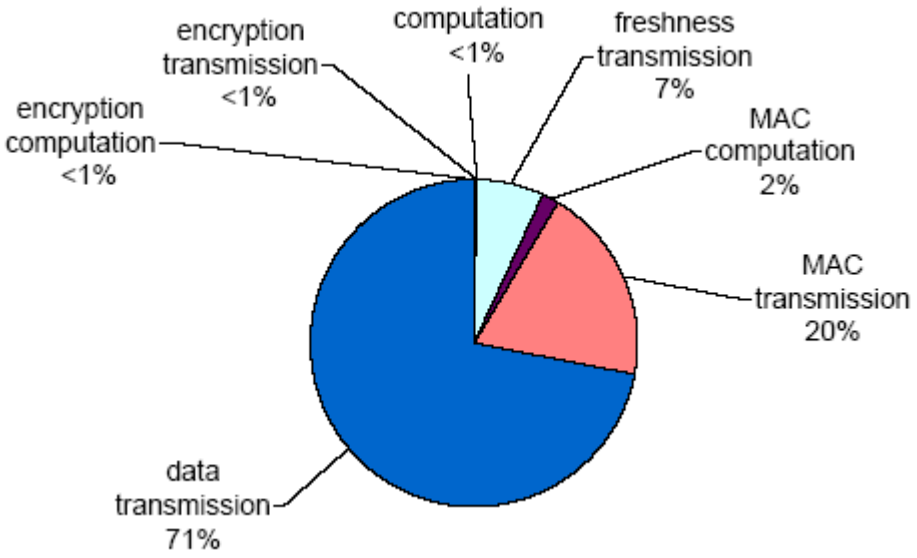


Figure 9.11: Energy costs of adding security protocols to the sensor network. Most of the overhead arises from the transmission of extra data rather than from any computational costs [16].

10. TINYSEC: A LINK LAYER SECURITY ARCHITECTURE FOR WIRELESS SENSOR NETWORKS

10.1 Introduction

"*TinySec is the first fully-implemented link layer security architecture for wireless sensor networks which is designed for resource constrained devices with parameter choices to find an optimal solution in terms of security, packet overhead, and resource requirements*" [19]. An optimized solution for performance does not mean extremely high security. TinySec is said to be designed from this principle to work with minimal overhead. If performance is degraded too much by an algorithm, then users will not turn on the security.

TinySec is designed as a plug-in replacement for existing insecure radio stacks, so cryptography can be enabled without requiring any changes to existing applications. The implementation is an extension to TinyOS, an open source operating system for sensor networks that is in widespread use.

10.2 Sensor Networks

There a lot of work in sensor design, both for hardware and software. For the proposed protocol the Mica2 mote and the TinyOS operating system is targeted which are both developed at the University of California at Berkeley.

The Mica2 mote is a small sensor unit with 8 MHz 8-bit Atmel ATMEGA128L CPU, a 433 MHz low-power radio from Chipcon, a power source, and several optional sensing elements. The processor used has 128 KB of instruction memory, 4 KB of RAM for data, and 512 KB of flash memory (Figure 10.1). The CPU consumes 4 mA (at 3 volts) when active, and two orders of magnitude less power when sleeping. The Chip-con radio consumes 8 mA (at 3 volts) in receive mode and up to 15 mA in transmit mode.

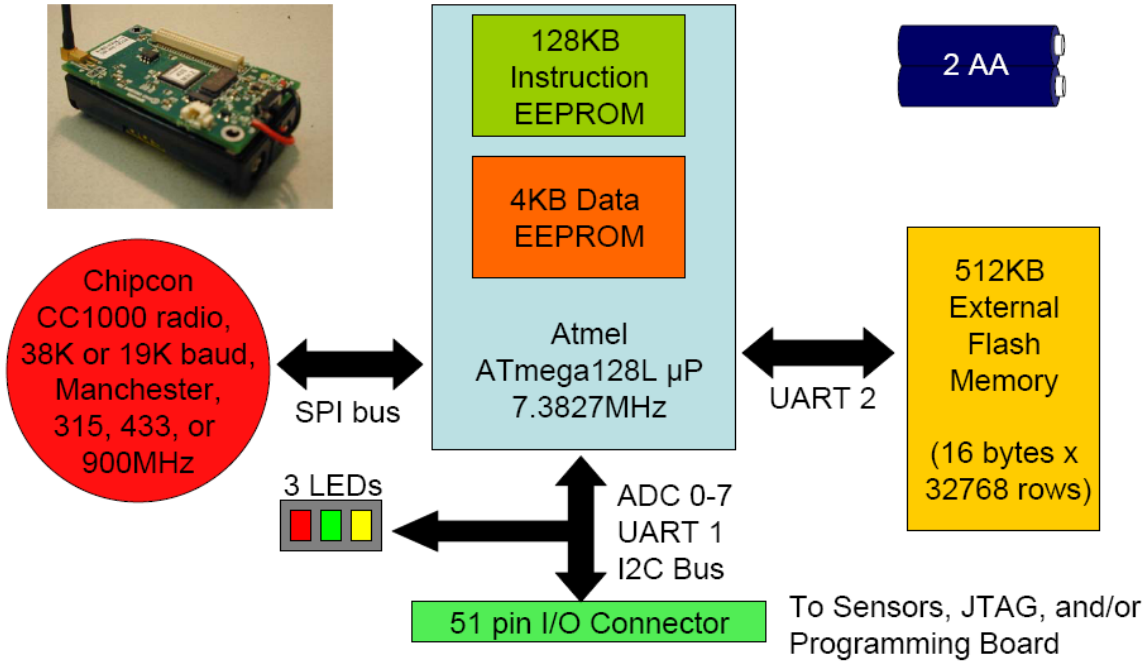


Figure 10.1: The Mica2 Mote [20]

The device has a power unit of two AA batteries, which provide approximately 2850 mA hours at 3 volts. At full power, the Mica2 mote can run for only two weeks. The motes run TinyOS which is an event-driven operating system for networked applications in wireless systems. Its memory requirements and code size is so small which makes it suitable for the devices with very little computational power but it is obvious that public-key cryptography is not usable with these systems.

"*In conventional networks, message authenticity, integrity, and confidentiality are usually achieved by an end to end security mechanism like SSH or SSL, but this is not the case in sensor networks. The traffic in sensor networks is usually many to one where sensor nodes send sensor readings or events over a multihop topology to a base station*" [19]. It is also stated that neighboring nodes often sense the same events and send the same packets to the base station which causes precious energy and bandwidth wastage.

It is suggested to use in-network processing where intermediate nodes access, modify, and suppress the contents of packets to solve this issue and it is obvious that end-to-end security mechanisms can not be used. Because of these reasons, link layer security architecture for TinySec is proposed where guaranteeing the authenticity, integrity, and confidentiality of messages between neighboring nodes.

10.3 Security

10.3.1 Security Goals

It is stated that a link layer security protocol should satisfy access control, message integrity, and message confidentiality. Access control means the link layer protocol should prevent unauthorized parties from participating in the network. The nodes should be able to detect the messages that are originated from unauthorized nodes and reject them where this is achieved using message authentication. Only the authenticated messages will be accepted, and to generate an authenticated message an appropriate cryptographic key should be known.

The system should be reactive to replay attacks. For an adversary eavesdropping on a secured message sent between two authorized nodes, it is possible to make this attack by replaying the packet at some later time which causes the receiver accept it again. Replay attacks seem to violate access control, but they do not. For defending replay attacks there can be an increasing counter with every message to check message freshness where the receiver can reject messages with old counter values. It is stated that the application layer can be better suited to check replay attacks where the TinySec does not protect against it at the link layer [19].

Message integrity is also an important issue which should be detected by the receiver, where an adversary modifies the message from an authorized sender while the message is in transit. The algorithm provides message integrity and authentication by applying a message authentication code to each packet. Confidentiality which means keeping the information secret from unauthorized parties can be achieved by using encryption.

10.3.2 Security Primitives

10.3.2.1 Message Authentication Codes (MACs)

For achieving message integrity and authenticity TinySec uses a message authentication code (MAC) which can be viewed as a cryptographically secure checksum of a message. It is stated that for computing a MAC authorized senders and receivers should share a secret key, and by using this key which is part of the input to a MAC computation, the sender computes a MAC over the packet by appending it to the packet.

The receiver sharing the same secret key computes the MAC of the received packet and compares it with the received MAC value. If they are equal, the receiver accepts the packet and rejects it otherwise. "*MACs should be hard to forge without the secret key to prevent an adversary computing the MAC value of an altered valid message or a bogus message. These messages will be rejected by the authorized receivers*" [19].

10.3.2.2 Initialization Vectors (IVs)

Semantic security can be achieved by using a unique initialization vector (IV) for each invocation of the encryption algorithm which causes to produce two different ciphertexts when the same plaintext is encrypted two times. If the encryption process is identical for two invocations on the same message, then semantic security is clearly violated. IV can be thought as a side input to the encryption algorithm which adds variation to the encryption process when there is little variation in the set of messages. IVs are generally included in the same packet with the encrypted data clearly.

10.4 Design of TinySec

For the network protocols such as IPSec, SSL/TLS, and SSH, it can be said that they are satisfactory for securing internet communications. However, these protocols are too heavy for securing sensor networks. It is known that these protocols add many bytes of overhead to the packets, and they need complex computations.

For some of the wireless and cellular systems, there are developed schemes that can be used in sensor networks, but these existing designs have serious limitations. For instance, the GSM frame format can protect voice data with little overhead, but researchers have found serious vulnerabilities with this security mechanism. Also there is WEP which is designed for 802.11 wireless networks, using RC4 encryption for confidentiality and a CRC checksum for integrity. However, WEP is found to be thoroughly flawed by many researchers with its short 24-bit IVs and the CRC checksum which fails to protect integrity.

There is also 802.11i's CCMP, which uses AES in COM mode, 48-bit IVs, and a strong 64-bit message authentication code which is well-designed, but the per-packet overhead is too high to be used in sensor networks.

10.4.1 Approaches

An IV (initialization vector) as an extra parameter to the encryption process should be used to support semantic security. The receivers and senders can maintain counters, synchronized implicitly at both ends which is too expensive because of the need to invoke a special re-synchronization procedure. Therefore, for TinySec IVs are used where the sender chooses an IV for each packet and transmits this IV in the packet.

The length of the IV, and the way of generation of IVs, has great effect on both security and performance. If the IV is too long, there will be unnecessary bits added to the packet and if it is too short the security warranty can not be supplied because of the IV repetition.

According to the pigeonhole principle it is said that an n-bit IV repeats after $2^n + l$ packets are sent but for some IV generation strategies, repetitions can occur earlier. "*For a random n-bit IV, it is expected to see the first repetition after* $2^{n/2}$ *packets have been sent according to the birthday paradox*" [19]. The proposed protocol uses a counter in the IV, and the counter is transmitted in the packet to the receiver.

A stream cipher uses the key and IV as a seed and stretches it into a large key stream which is then XORed against the message. Stream ciphers are faster than the block ciphers but it fails if the same IV is ever used to encrypt two different packets which make the recovery of both plaintexts possible.

A block cipher is known as a keyed pseudorandom permutation over small bit strings, typically 8 or 16 bytes like DES, AES, RC5, or Skipjack. If messages are longer than 8 or 16 bytes, block ciphers require a mode of operation to encrypt them.

Using a block cipher for encryption has an additional advantage and the most efficient message authentication code (MAC) algorithms also use a block cipher. So, using the same block cipher algorithm for both encryption and MAC saves on code space.

To use block ciphers for encryption, a mode of operation should be chosen. For CBC mode if two different plaintexts P and P' is encrypted then the cryptanalyst learns nothing from these cipher texts. CBC mode works with a random IV, and has some problems when used with a counter as IV. If two plaintexts P and P' is encrypted by using IV and IV' respectively, and $P_1 \oplus IV = P_1' \oplus IV'$, then the first block of ciphertexts will be equal. This discloses the value $P_1 \oplus P_1'$. For instance, if IV is a

counter then IV' = IV ⊕ 1. And for the plaintexts if P' = P ⊕ 1, then there appears leakage which is said to be undesirable [19].

Figure 10.2 shows CBC mode, as used in TinySec for encryption. The IV is preencrypted where it is not used in the standard CBC mode. E_k is a block cipher with a key k, M_j represents the i-th block of the message, and IV is the initialization vector.

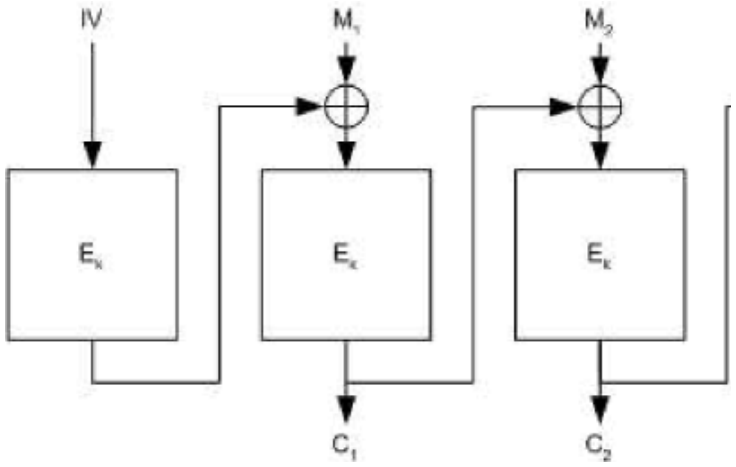


Figure 10.2: CBC mode for encryption [19].

For using CBC mode with non-repeating IV, pre-encrypting the IV is suggested. It is stated that when a block cipher is needed, either AES or Triple-DES should be used. But both AES and Triple-DES are unsuitable for implementation in embedded systems because of computation power and speed issues. Figure 10.3 shows the CBC-MAC which is used in TinySec for message authentication. In this mode the length is prepended to the message.

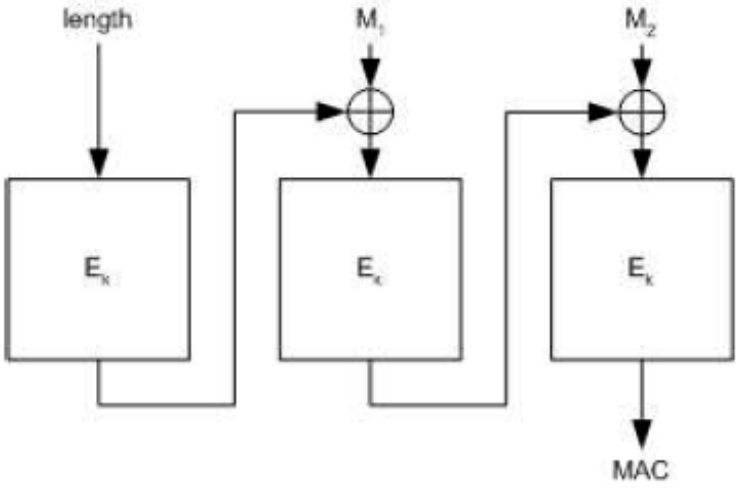


Figure 10.3: CBC-MAC for authentication [19].

It is stated that the counter mode for encryption is not chosen for mode of operation because it is a stream cipher mode of operation and it shares all the problems of any other stream cipher (Figure 10.4).

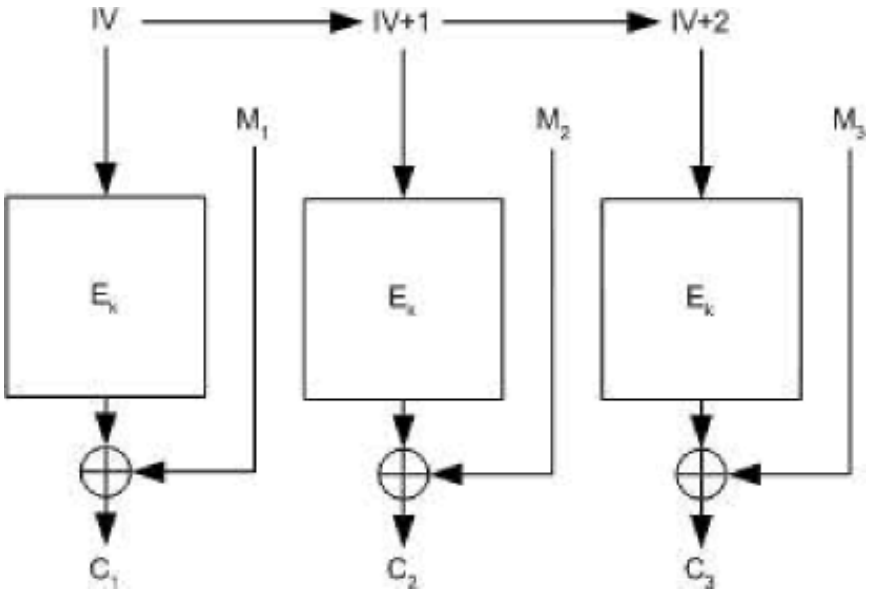


Figure 10.4: Counter (CTR) mode for encryption [19].

For TinySec implementation RC5 can also be selected, but RC5 has some disadvantages. It is patented, so it is not suitable for widespread deployment. It also needs a slow key setup stage, where the 128-bit RC5 key is expanded to a 104-byte expanded key. After that another alternative Skipjack algorithm is chosen. Skipjack uses 80-bit key without key setup stage which does not require expanded keys, and is unpatented [19].

10.4.2 TinySec Design and Packet format

TinySec supports two different security options: authenticated encryption (TinySec AE) and authentication only (TinySec Auth). It is stated that with authenticated encryption, TinySec encrypts the data payload and authenticates the packet with a MAC where MAC is computed over the encrypted data and the packet header. In authentication only mode, TinySec authenticates the entire packet with a MAC, but the data payload is not encrypted.

In TinySec authenticating messages is mandatory, but encryption is optional. In some cases encryption is not needed and if it is used latency is increased, more computation and battery power is needed. Nearly all applications require packet authenticity to drop invalid messages injected by an adversary.

As TinySec is designed on TinyOS, its packet format depends on the packet format of TinyOS. It is stated that the common fields of both packets are destination address, active message (AM) type (like port number), and length where all these fields are not encrypted for saving power (Figure 10.5) [19].

Figure 10.5: TinyOS packet format [19].

Sensor nodes can determine that the message is not addressed to it and turn off its radio which can be an early detection. Also encrypting the length field is not so important because the length of message can be identified easily.

TinyOS uses 16-bit cycle redundancy check (CRC) to detect transmission errors but CRCs are not secure and can be computed easily. TinySec replaces the CRC with MAC for message integrity, authenticity and detection of transmission errors. There is a group field in TinyOS packet format which is used to differ sensor networks from each other but TinySec does not use this field as it is said to be unnecessary (Figure 10.6).

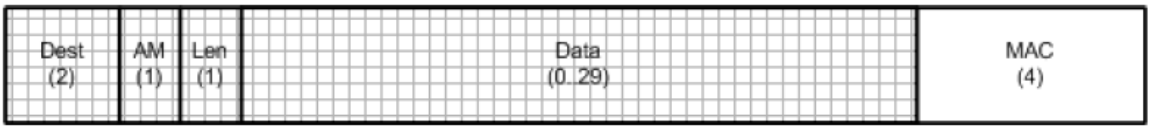


Figure 10.6: TinySec-Auth packet format [19].

TinySec uses a special 8 byte IV which is like “dest || AM || i || src || ctr”, where dest is the destination address of the receiver, AM is the AM type, i is the length of the data payload, src is the source address of the sender, and ctr is a 16 bit counter which is increased after each message sent (Figure 10.7).

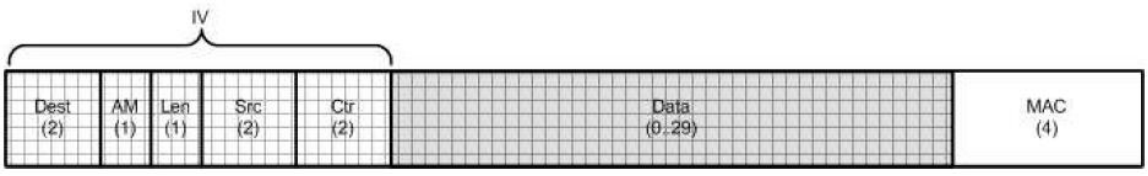


Figure 10.7: TinySec-AE packet format [19].

10.4.3 Encryption and Authentication Primitives

For the implementation of TinySec CBC-MAC is chosen because of its efficiency, speed and the minimized number of cryptographic primitives. CBC-MAC is said to be not secure for variable sized messages where rivals can attack by using a MAC for certain messages.

"*For computing the MAC in addition to data payload, the destination address, AM type, length, source address and counter is used which helps to prevent attackers from changing the packet address, truncating the packets and some other attacks*" [19].

10.4.4 Keying Mechanisms

TinySec uses two keys for encrypting data and computing MACs. TinySec uses a very simple, globally shared keying mechanism and this single globally shared key makes the system very usable and easily configurable. Within this system two authorized nodes can communicate with each other securely and unauthorized nodes can not send any packet to the network where all of its packets are rejected.

In TinySec design key distribution is not very good because shared key is loaded to the nodes before deployment. The key can not be changed on the run time because the key is set in compile time and loaded to the devices. TinySec does not currently support very efficient keying mechanisms but it is suggested for a node to use a key for communicating with only its neighbors where a node can only decrypt the packets addressed to it and can only send packets to its neighbors. In fact TinySec protocol is not limited to any particular keying mechanism and any mechanism can be used in conjunction with TinySec. There exists also some research for the keying mechanism of TinySec. They are located in the CVS repository of the TinyOS operating system source files.

"*Normally a keying mechanism depends on some factors like easily usage and the networking and security requirements of applications. Different keys for different applications can also be used. The simplest approach is using single network wide key among all nodes which maximizes usability, and minimizes configuration*" [19].

Any of the two nodes can easily communicate with each other by using these keys and the shared key is loaded into the nodes before deployment as it is being done in the current implementation. If someone learns the secret key, she can eavesdrop on traffic and inject messages anywhere in the network.

Another option is using a shared key for communication only if two nodes need to communicate with each other. A separate TinySec key for each pair of nodes can be used to provide better protection against node capture attacks but key distribution and local broadcasting issues are not easy.

Another option is sharing a different TinySec key for groups of neighboring nodes which does not prevent local broadcast but key distribution is still not easy. A compromised node can decrypt messages only from its group and can not decrypt the messages of other groups.

10.5 Security Analysis

10.5.1 Message Integrity and Authenticity

The length of MAC is 8 or 16 bytes in most of the today's protocols. The security of a k-bit MAC is identified by comparing to a random selected function from the set of all functions. For example if CBC-MAC is used as a random function with 4 bytes of output, then an attacker has probability of $1/2^{32}$ for finding a valid MAC for a particular message.

It is stated that for a network using TinySec finding a valid MAC with a 19.2 Kb/s channel (40 packets per second) requires over 3 years for an attacker. It is also easy to detect when such an attack is happening where nodes can warn the base station when the rate of MAC failures exceeds some predetermined value [19].

10.5.2 Confidentiality

The main property of IVs is to be unique, but using an 8 byte counter or 16 byte random value makes the IV easily repeatable. TinySec uses an 8 byte IV where the first 4 bytes are generated from the destination address, the AM type, and the length.

The other 4 bytes of the IV are generated from src which is the source address of the sender and ctr which is a 16 bit counter starting at 0 (Figure 10.7).

The format for the last 4 bytes guarantees that each node can send at least 2^{16} (65,000) packets before IV repetition occurs. It is stated that if there are n nodes sending packets at the same rate then IV reuse can occur after $n \bullet 2^{16}$ packets are sent in the network according to the birthday paradox. For a sensor node sending one packet per hour makes the system work for over 7 years without IV reuse.

The algorithm selected CBC mode because of its robustness to information leakage when IVs repeat. It is proposed that in stream ciphers a repeated IV can reveal the plaintext of both messages, but in CBC mode IV reuse reveals only the length of the longest shared prefix of the two messages. It is obvious that the dest||AM||l which forms the first 4 bytes of the IV help to prevent information leakage when the counter repeats on a node except for the dest||AM||l values to be exactly the same for both messages.

Information leakage only occurs if a node sends two different packets with the same first 8 bytes and IV, to the same destination, with the same AM type, and of the same length.

10.6 Implementation

TinySec is implemented for the Berkeley sensor network motes on TinyOS and it is an open source project within the TinyOS mainline supporting the Mica, Mica2, and Mica2Dot platforms. The Mica mote uses the RFM TR1000 radio, the Mica2 and Mica2Dot motes use the Chipcon CC1000 radio, but it is easily portable to both new processors and new radio architectures.

TinySec has 3000 lines of nesC code which creates an overhead of 728 more bytes of RAM and 7146 bytes more program space on an application using TinySec's first implementation. Also the static lookup table of the Skipjack algorithm can be in RAM or program space which makes the algorithm 10% slower and as the motes have available RAM, the static table is stored in RAM.

"*For the current TinySec release, it is optimized to use 256 bytes of RAM and 8152 bytes of ROM. The optimizations save 472 bytes of RAM at the expense of 6% slower block cipher operations*" [19].

TinyOS radio stack is changed to create TinySec and a new secure radio component is developed for the supported platforms. For example instead of using TinyOS default radio component, TinySec's secure radio component is tied to the application. This is supplied by setting the "TINYSEC" parameter in the makefile of your application to true. TinyOS build scripts fetch the correct components for the TinySec and use them (see Figure 10.8).

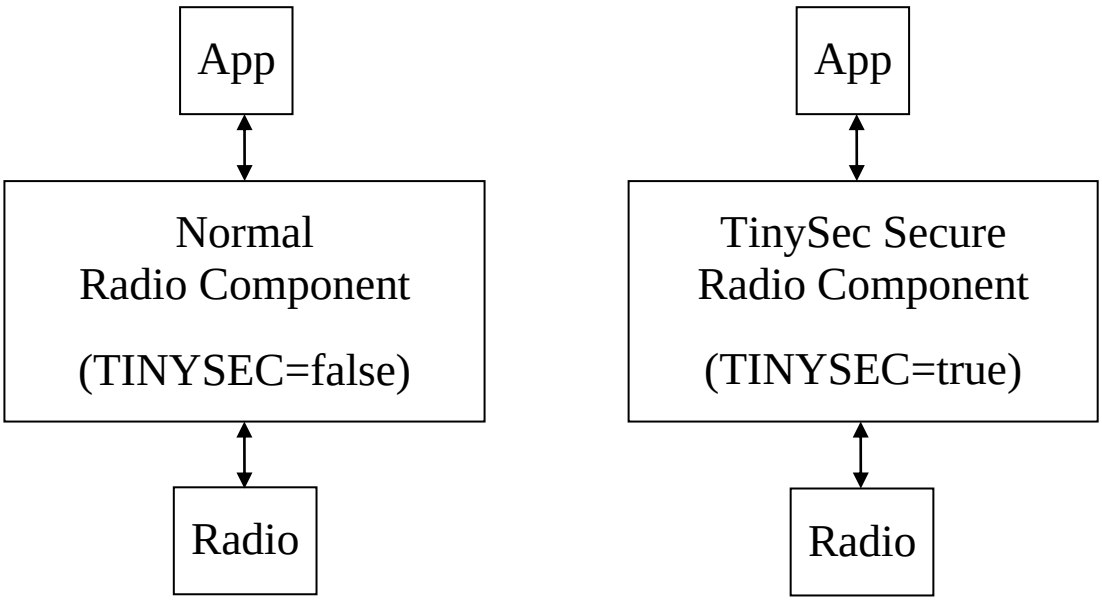


Figure 10.8: TinySec and TinyOS interfaces [20].

The TinySec protection level is stored in the upper two bits of the length byte because the maximum data payload is 32 bytes (TinySec AE, TinySec Auth, or normal TinyOS packets).

For the development of TinySec, TinyOS scheduler is changed to complete cryptographic computations on time. When the radio channel is acquired TinySecM module is started and the cryptographic computations are done.

This process must be completed by the time the radio finishes sending the start symbol. "*The TinyOS scheduling process which runs tasks in FIFO order is modified to include two priority schedulers, where cryptographic operations are run with high priority where the computations are performed as the radio stack receives enough data*" [19].

Another valuable property of TinySec is it is cipher independent and it can use Skipjack with the current implementation. Other algorithms can also be adopted to be used by TinySec easily.

10.7 Measurements

It is obvious that using TinySec has some computational and energy costs which are the larger packet sizes and the extra computation time and energy that is needed for cryptographic computations. The first parameter that is calculated is the effect of

packet lengths where TinySec increases packet lengths by 1 or 5 bytes for TinySec Auth or TinySec AE usage respectively.

It is stated that some tests have been done by using TinySec-AE and it is seen that the packet latencies are increased by 8.0% (compared to the current TinyOS stack) and the tests that are done by using TinySec-Auth has shown that it causes a 1.5% latency increase (Table 10.1).

Table 10.1: Table listing the expected overhead costs using TinySec for a 24 byte data payload. The packet overhead includes space needed for the header and media access control information. Since TinySec increases the packet size by a fixed amount, it will increase the time needed to send the packet over the radio. This impacts bandwidth, latency, and the energy needed to send a packet [19].

	Application Data (b)	Packet Overhead (b)	Total Size (b)	Time to Transmit (ms)	Increase Over Current TinyOS Stack
Current TinyOS Stack	24	39	63	26.2	—
TinySec-Auth	24	40	64	26.6	1.5%
TinySec-AE	24	44	68	28.8	8.0%

To measure TinySec performance costs the impact of TinySec on bandwidth, energy, and latency on the Berkeley Mica2 sensor nodes are considered. The term “byte time” is the time to transmit a single byte over the radio which is directly related to the packet transmission time.

Figure 10.9 shows the result of the measurements for end to end latency in a system using TinySec. It is stated that 37 nodes are used for this test and the time it took to send a message over some number of hops is calculated. It is seen that TinySec increases the message reception latency because it increases the packet lengths and longer packets take longer time to transmit.

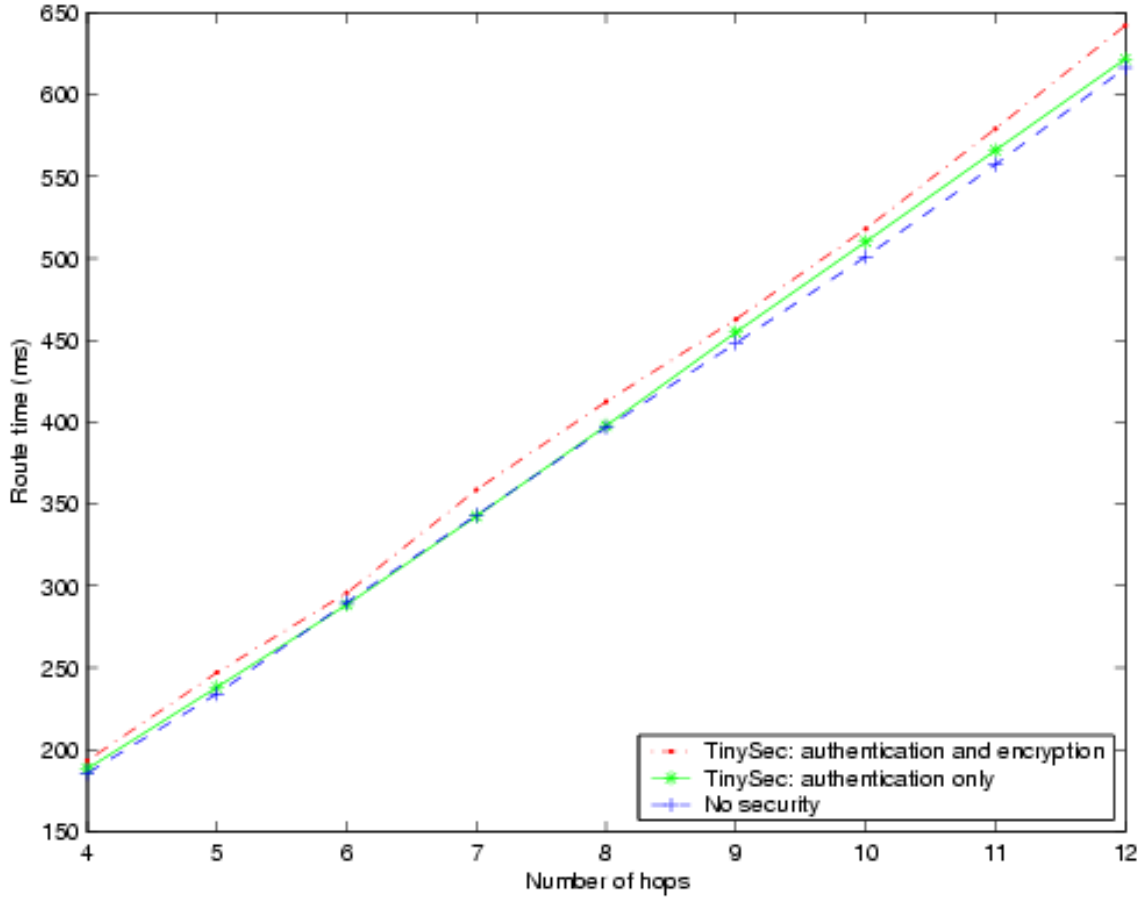


Figure 10.9: End to end latency measurements [19].

As the TinySec authentication only feature increases the packet lengths only 1 byte, its latency is nearly the same where no security is used. However if TinySec AE is used message latency increases.

Figure 10.10 shows the increase in latency when sending packets in TinySec Auth and TinySec AE modes. It is seen that if TinySec AE mode is used latency is increased by 4.6 byte times. As TinySec AE adds extra 5 bytes to the packets the result is consistent. If encryption is not used and the packets are sent in TinySec Auth mode theoretically 1 byte time should be observed and according to the measurements 1.1 byte time latency increase is observed.

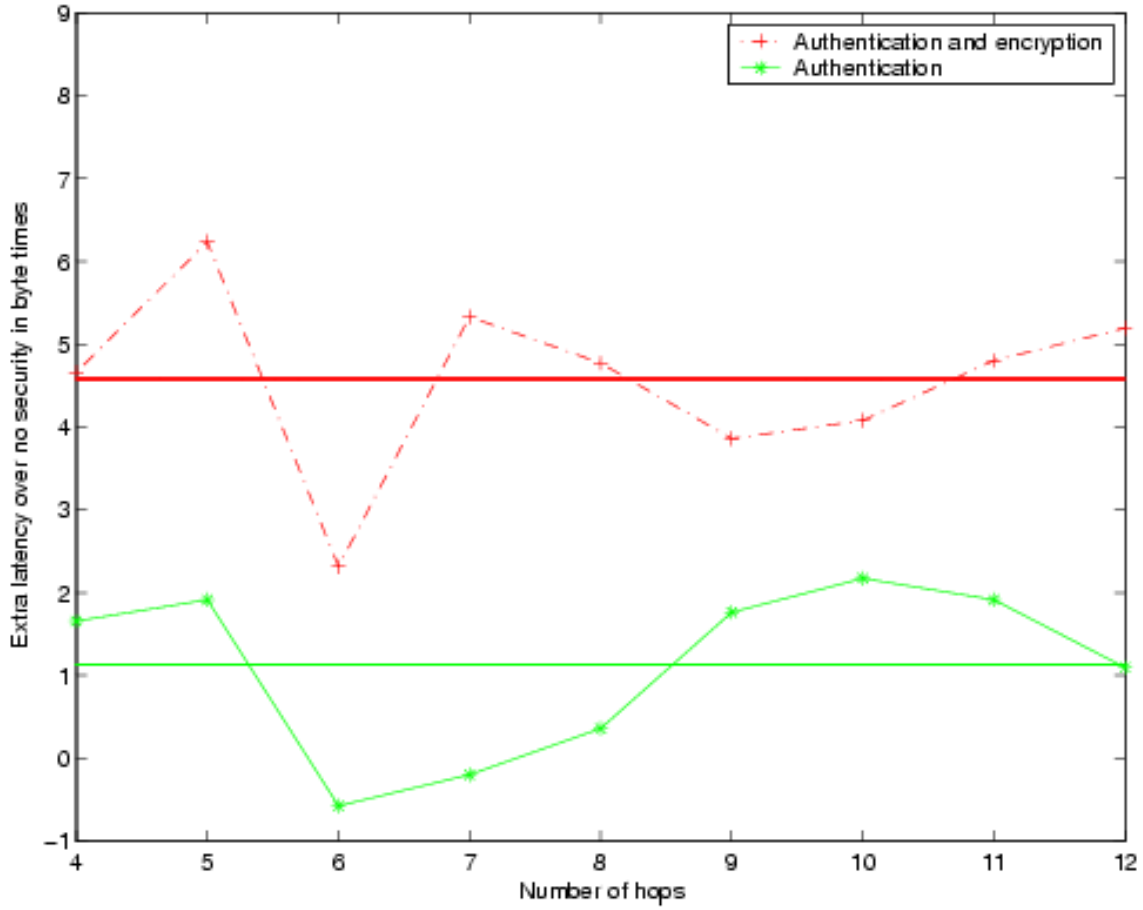


Figure 10.10: Increase in latency when TinySec is used (byte times) [19].

For testing the consumed energy a single mote running a simple application that sends packets is used. The consumed energy for sending a packet with the default stack is calculated as 0.00016 mAH by using an oscilloscope. If packets are transmitted in TinySec Auth mode the consumed energy is calculated as 0.000165 mAH, and if TinySec-AE mode is used to transmit the sample packets the consumed energy is increased to 0.000176 mAH (Table 10.2).

Table 10.2: Total energy consumed to send a 24 byte packet.

	Energy (mAH)	Increase
Current TinyOS Stack	0.000160	—
TinySec-Auth	0.000165	3%
TinySec-AE	0.000176	10%

Figure 10.11 shows the current in amperes with respect to time. It is seen that using TinySec authentication only feature increases energy consumption by 3% and if the TinySec authentication and encryption feature is used the increase in energy consumption is 10%. It is stated that for TinySec Auth 1% of 3% energy overhead comes from increased packet length and 2% from extra cryptographic computations. For the TinySec AE mode 6% of 10% energy overhead comes from increased packet length and 4% from extra cryptographic computations.

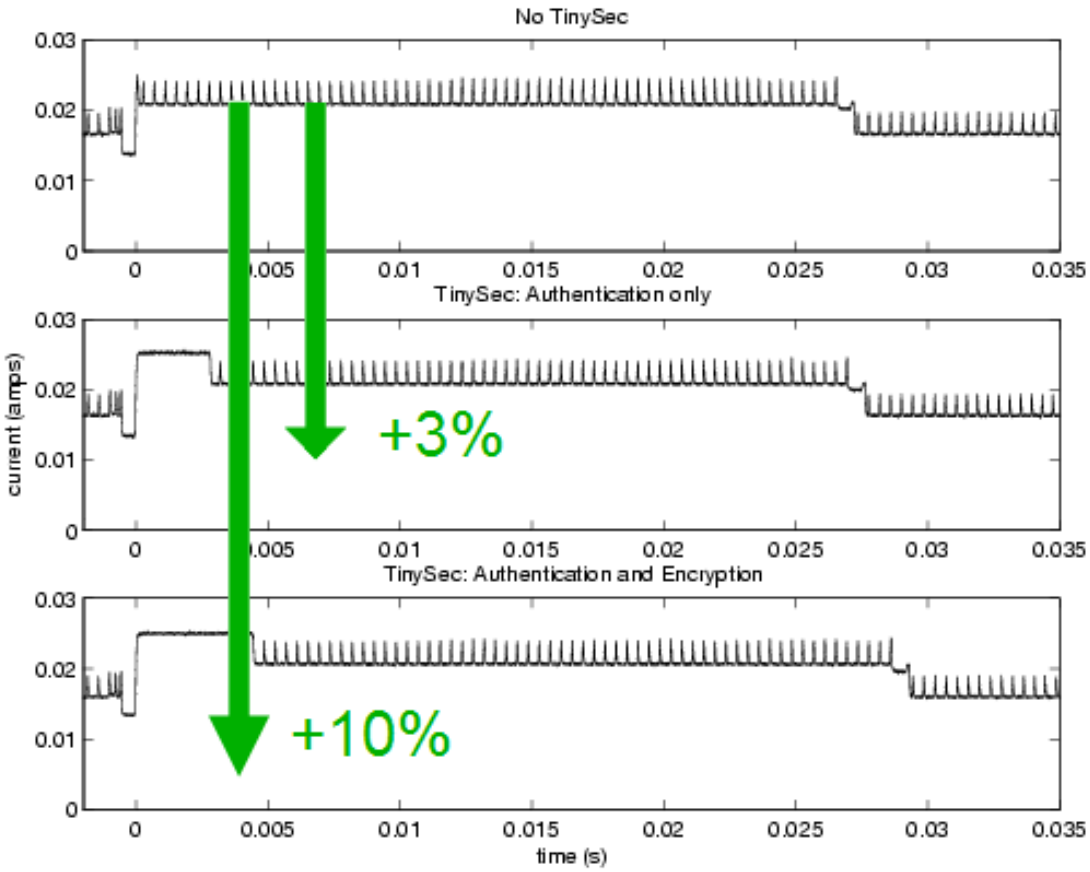


Figure 10.11: The power consumption for sending a packet [19].

Another performance measure is done on the maximum throughput as the total number of packets that could be sent successfully in 30 seconds with variable number of senders and 24 bytes of application data. TinySec-Auth's bandwidth is very close the current TinyOS stack but TinySec-AE causes a bandwidth decrease by 6% which sends 5 byte larger packets (Figure 10.12). Throughput difference seems to be only due to differences in packet length, not the computational costs.

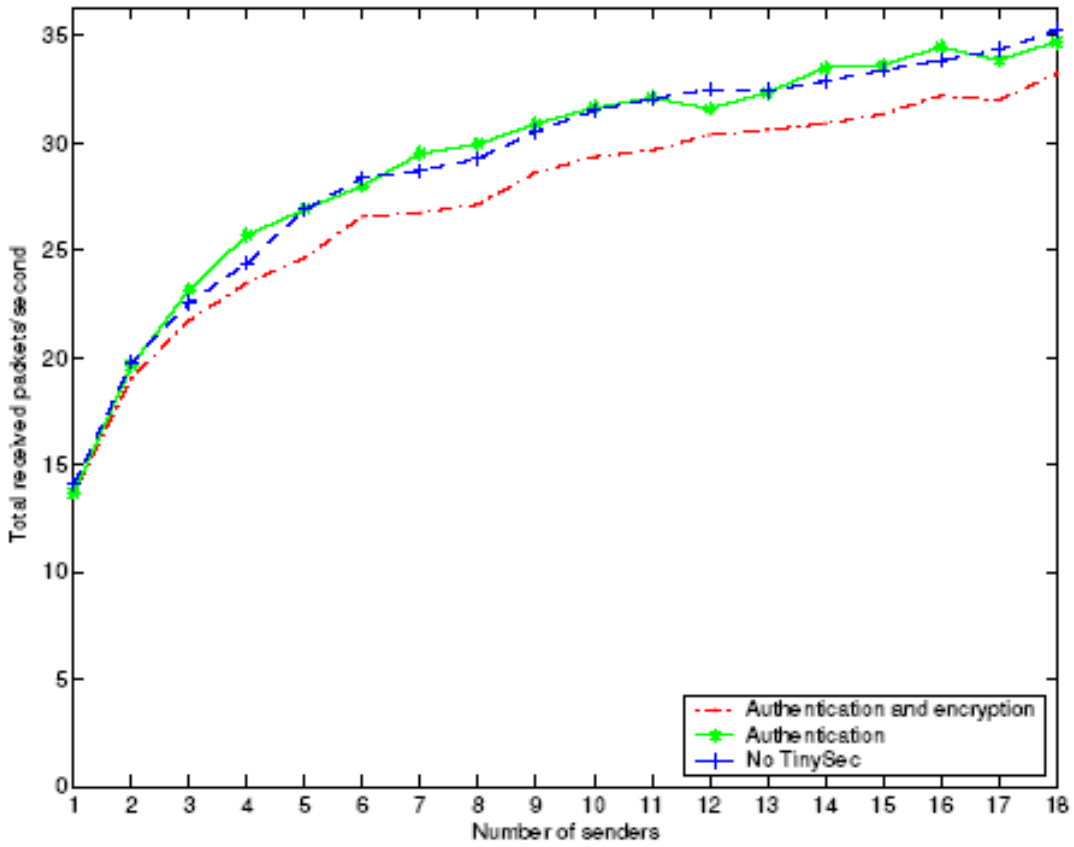


Figure 10.12: Bandwidth, plotted as a function of the number of send-receive pairs. We compare TinySec with and without authentication to the bandwidth without using TinySec [19].

It is also computed that routing with TinySec-Auth takes more time than the routing with the current TinyOS radio stack and additionally routing with TinySec-AE takes more time than both of them.

It is seen that the energy, bandwidth, and latency overhead of TinySec is less than 10% where most of the overhead is caused by the increased packet length. "*It is also obtained that the additional energy cost for the cryptographic computations is 2/3 of the total energy increase of TinySec-AE*" [19].

11. RELATED WORK

11.1 Underlying Encryption and Decryption Algorithms

TinySec has a flexible architecture that can use any encryption and decryption algorithm. Algorithms like AES, DES, RC5 or Skipjack can easily be adopted to be used with TinySec.

11.1.1 Skipjack

SKIPJACK is a 64-bit electronic codebook algorithm that transforms a 64-bit input block into a 64-bit output block. The transformation is parameterized by an 80-bit key, and involves performing 32 steps or iterations of a complex, nonlinear function. The algorithm can be used in any one of the four operating modes defined in FIPS 81 for use with the Data Encryption Standard (DES) [21].

The SKIPJACK algorithm was developed by NSA and is classified secret. It is representative of a family of encryption algorithms developed in 1980 as part of the NSA suite of "Type I" algorithms, suitable for protecting all levels of classified data. The specific algorithm, SKIPJACK, is intended to be used with sensitive but unclassified information.

The strength of any encryption algorithm depends on its ability to withstand an attack aimed at determining either the key or the unencrypted plaintext communications. There are basically two types of attack, brute-force and shortcut.

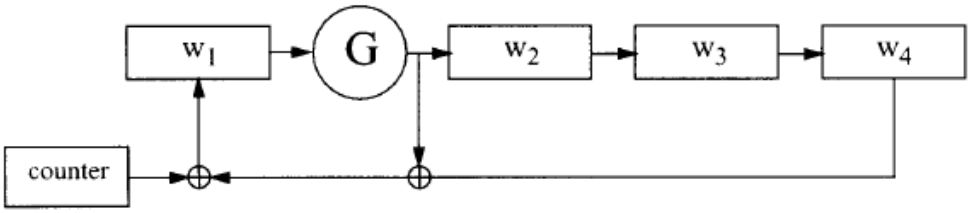
The decision not to make the details of the algorithm publicly available has been widely criticized. Many people are suspicious that Skipjack is not secure, either due to oversight by its designers, or by the deliberate introduction of a secret trapdoor.

Aware of such criticism, the government invited a small group of independent cryptographers to examine the Skipjack algorithm. They issued a report which stated that, although their study was too limited to reach a definitive conclusion, they believed that Skipjack was secure. Another consequence of Skipjack's classified

status is that it cannot be implemented in software, but only in hardware by government authorized chip manufacturers [21].

Skipjack encrypts 4 word data blocks by alternating between two stepping rules which can be seen figure 11.1 and figure 11.2. For the rule A the following rules apply :

- *"G permutes w_1,*
- *The new w_1 is the xor of the G output, the counter and w_4,*
- *Words w_2 and w_3 shift one register to the right and become w_3 and w_4 respectively,*
- *The new w2 is the G output,*
- *The counter is incremented by one"* [21].



Rule A

$$w_1^{k+1} = G^k(w_1^k) \oplus w_4^k \oplus counter^k$$
$$w_2^{k+1} = G^k(w_1^k)$$
$$w_3^{k+1} = w_2^k$$
$$w_4^{k+1} = w_3^k$$

Rule A^{-1}

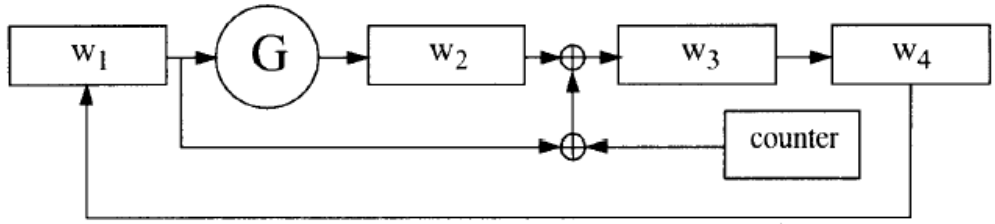
$$w_1^{k-1} = [G^{k-1}]^{-1}(w_2^k)$$
$$w_2^{k-1} = w_3^k$$
$$w_3^{k-1} = w_4^k$$
$$w_4^{k-1} = w_1^k \oplus w_2^k \oplus counter^{k-1}$$

Figure 11.1: Rule A for the Skipjack algorithm [21].

For the rule B similarly the following rules apply :

- *"G permutes w_1,*
- *The new w_1 is w_4,*

- *The new w_3 is the xor of w_2, counter and w_1,*
- *The new w2 is the G output,*
- *The new w_4 is w_3*
- *The counter is incremented by one"* [21].



Rule B

$$w_1^{k+1} = w_4^k$$
$$w_2^{k+1} = G^k(w_1^k)$$
$$w_3^{k+1} = w_1^k \oplus w_2^k \oplus counter^k$$
$$w_4^{k+1} = w_3^k$$

Rule B^{-1}

$$w_1^{k-1} = [G^{k-1}]^{-1}(w_2^k)$$
$$w_2^{k-1} = [G^{k-1}]^{-1}(w_2^k) \oplus w_3^k \oplus counter^{k-1}$$
$$w_3^{k-1} = w_4^k$$
$$w_4^{k-1} = w_1^k$$

Figure 11.2: Rule B for the Skipjack algorithm [21].

The algorithm requires 32 steps to complete encryption and decryption. For encryption the counter is initialized to 1 and Rule A is applied 8 times, incrementing the counter at each step. After that the algorithm switches to Rule B and applies it 8 times. Then it returns to Rule A, after applying 8 steps finally Rule B is applied 8 times again.

For the decryption the counter is set to 32 and Rule B^{-1} is applied 8 steps, then Rule A^{-1} is applied 8 steps. After that the algorithm switches back to Rule B^{-1} again, after applying it 8 times, finally Rule A^{-1} is applied 8 times to complete decryption [21].

In the Skipjack algorithm G is a crypto variable dependent permutation which uses a special substitution table. The crypto variable that is used by this permutation is 10 bytes long. G has a four round structure and at every round a substitution is done by a fixed byte substitution table. Each round one byte of the crypto variable is also used for substitution.

Figure 11.3 shows the G permutation and the inverse of G schematically. F is the fixed byte substitution table, cv_k is the kth byte of the crypto variable. As the crypto variable is 10 bytes long, mod 10 should be used.

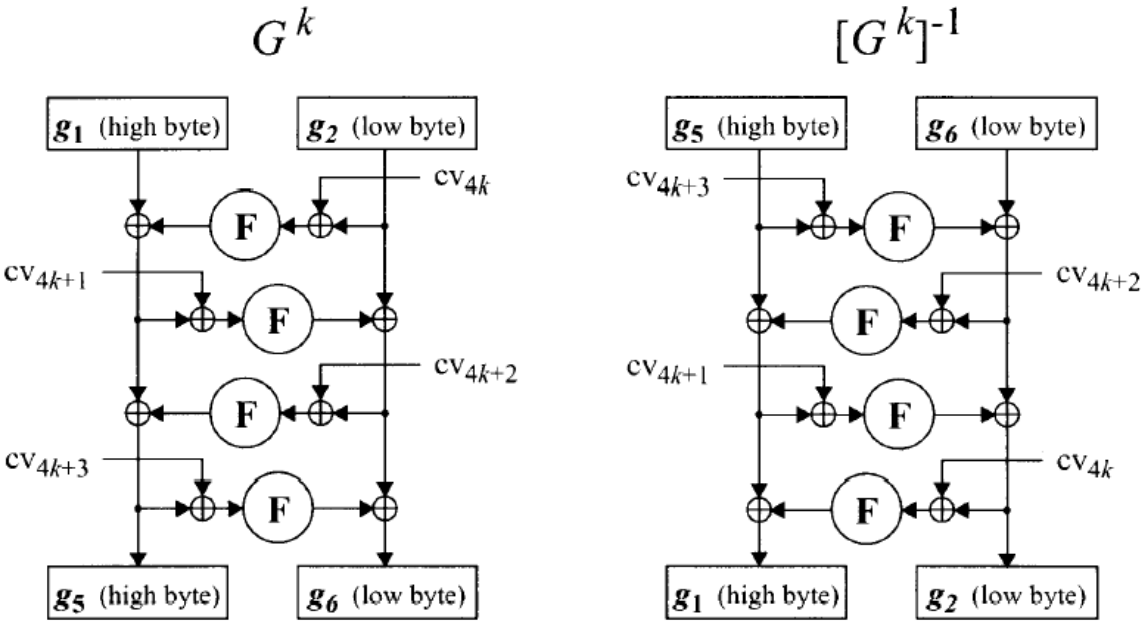


Figure 11.3: G permutation diagram [21].

11.1.2 RC5

RC5 [22] is a fast block cipher designed by Ronald Rivest for RSA Data Security (now RSA Security) in 1994. It is a parameterized algorithm with a variable block size, a variable key size, and a variable number of rounds. Allowable choices for the block size are 32 bits, 64 bits (replacement for DES), and 128 bits. The number of rounds can range from 0 to 255, while the key can range from 0 bits to 2040 bits in size. Such built-in variability provides flexibility at all levels of security and efficiency.

There are three routines in RC5: key expansion, encryption, and decryption. In the key-expansion routine, the user-provided secret key is expanded to fill a key table whose size depends on the number of rounds. The key table is then used in both encryption and decryption. This key setup comes with latency overhead and this is the disadvantage of this block cipher. The encryption routine consists of three primitive operations: integer addition, bitwise XOR, and variable rotation. The exceptional simplicity of RC5 makes it easy to implement and analyze.

"*The heavy use of data-dependent rotations and the mixture of different operations provide the security of RC5. In particular, the use of data-dependent rotations helps defeat differential and linear cryptanalysis*" [23].

RC5 is a word oriented cipher and it can be said that all of the primitive operations work on w bit words which is the basic unit of the algorithm. RC5 has a key table S derived from the user supplied secret key. The algorithm generates an expanded key table by using this secret key. The size t of table S depends on the number r of rounds. S has t=2(r+1) words.

RC5 algorithm has 3 adjustable parameters and as a result of this every RC5 algorithm is designated by using these parameters as RC5-w/r/b. "w" is the word size in bits and the standard value of the word size of the RC5 algorithm is 32 bits. "w" can be selected as 16, 32 or 64 bits. The word size w also affects speed and security. For example, choosing a value of w larger than the register size of the CPU can degrade encryption speed. So the selected word size should be equal to the register size of the CPU for embedded systems.

As RC5 encrypts two-word blocks, plaintext and ciphertext blocks are each 2w bits long. "r" is the number of rounds and the expanded key table contains 2(r+1) words. The value of r should be between 0 and 255. RC5 algorithm's last adjustable parameter b is the length of the secret cryptographic key (K) which can be between 0 and 255 [22]. The parameters of the algorithm should be selected according to the application's security needs.

"*For example, RC5-32/16/10 has 32 bit words, 16 rounds, a 10 byte (80 bit) secret key variable, and an expanded key table of 2(16+1)=34 words*" [22].

For expanding the user supplied b byte secret key, first the key is converted from byte sized array to word sized array.

```
for i = b - 1 downto 0 do

    L[i/u] = (L[i/u] <<< 8) + K[i];
```

K[i] is the ith byte of the user supplied key, L is the new array of size c=b/u and u=w/8. "+" operation is two's complement addition of words which is modulo 2^w addition. The inverse operation subtraction is denoted "-".

The rotation of word x left by y bits is denoted x <<< y. Only the $\log_2 w$ least significant bits of y are used to determine the rotation amount and the inverse operation right rotation is denoted as x >>> y.

Then the table S is initialized by using two word-sized binary constants which are called magic constants (P_w and Q_w). These constants are dependant to the selected word size of the algorithm.

$S[0] = P_w$;

for i = 1 to t - 1 do

$S[i] = S[i - 1] + Q_w$;

"*The next step of key expansion is mixing the user's secret key in three passes over the arrays S and L and due to the differing sizes of these arrays the largest array will be processed three times and the other may be handled more times*" [22].

i = j = 0;

A = B = 0;

do 3 * max(t, c) times:

A = S[i] = (S[i] + A + B) <<< 3;

B = L[j] = (L[j] + A + B) <<< (A + B);

i = (i + 1) mod(t);

j = (j + 1) mod(c);

Let (L_0, R_0) denote the left and right halves of the plaintext. Then the encryption algorithm is given by:

$L_0 = A + S[0]$;

$R_0 = B + S[1]$;

for i = 1 to r do

$L_i = ((L_{i-1} \text{ XOR } R_{i-1}) <<< R_{i-1}) + S[2 * i]$;

$R_i = ((R_{i-1} \text{ XOR } L_i) <<< L_i) + S[2 * i + 1]$;

The decryption algorithm is given by:

for i = r down to 1 do

$R_{i-1} = ((R_i - S[2*i+1])>>>L_i) \text{ XOR } L_i);$

$L_{i-1} = ((L_i - S[2*i] >>> R_{i-1}) \text{ XOR } R_{i-1});$

$B = R_0 - S[1];$

$A = L_0 - S[0];$

"*A distinguishing feature of RC5 is its heavy use of data dependent rotations. The amount of rotation performed is dependent on the input data and is not predetermined. The encryption and decryption routines are very simple. While other operations such as substitution operations could have been included in the basic round operations, the objective is to focus on the data dependent rotations as a source of cryptographic strength*" [22].

11.1.3 Differences between RC5 and Skipjack

Skipjack uses 80-bit key and RC5 has a variable key size. Skipjack involves performing 32 steps of a complex and nonlinear function, but RC5 is much simpler where it consists of three primitive operations: integer addition, bitwise XOR, and variable rotation. There is a key setup operation which causes latency on startup for RC5 where Skipjack has no latency on startup.

RC5 is easy to implement and analyze because of its simplicity. RC5 is a parameterized algorithm with a variable block size, a variable key size, and a variable number of rounds which makes it more flexible than Skipjack. There are crypto variable dependent permutations in Skipjack and there are data dependent rotations and the mixture of different operations in RC5. One disadvantage of RC5 is it is patented.

11.2 Using Skipjack and RC5 with TinySec for Authentication

TinySec uses the same algorithm for both encryption and authentication purposes. The performance changes will be calculated when this algorithm changes. The current TinySec uses Skipjack and for this work RC5 is adopted to be used for authentication.

For testing the performance of TinySec two 64-bit block ciphers, Skipjack and RC5 is used and compared. As it is stated earlier that cryptographic operations should be completed very quickly because they must be completed before the radio operations. If these operations do not complete in time, the data needed for the radio will not be available.

It can be seen that RC5 and Skipjack are pretty good algorithms where each block cipher operation takes less than a byte time. It is said that the block cipher operation should complete in under a few byte times to not encounter any problems because the radio is waiting for these cryptographic operations. Currently implemented TinySec uses Skipjack as the default algorithm because of its minimal key setup costs and patent free usage.

11.2.1 Simulation and Test Environment

For simulations and tests some applications that can be run on PC are used. The TinyOS applications can be compiled for both mica motes and for PC environment. When the application is compiled for PC it is simulated by a discrete event simulator which is called TOSSIM [24].

TOSSIM is a discrete event simulator for TinyOS sensor networks. Instead of compiling a TinyOS application for a mote, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment. As TOSSIM runs on a PC, users can examine their TinyOS code using debuggers and other development tools.

TOSSIM is automatically built when an application is compiled. Applications are compiled by entering an application directory and typing make. Alternatively, when in an application directory, typing "make pc" will only compile a simulation of the application [24]. This simulation application can be run directly on PC. The generated application can be run both on cygwin environment (.exe file) and on Linux.

TOSSIM can simulate thousands of nodes simultaneously. Every mote in a simulation runs the same TinyOS program.

There is another special user interface for TOSSIM which is called TinyViz. TinyViz is a Java visualization and actuation environment for TOSSIM. The main TinyViz class is a jar file, tools/java/net/tinyos/sim/tinyviz.jar. TinyViz can be attached to a running simulation. Also, TOSSIM can be made to wait for TinyViz to connect

before it starts up, with the -gui flag. This allows users to be sure that TinyViz captures all of the events in a given simulation. TinyViz is not actually a visualizer, it is a framework in which plugins can provide desired functionality. TinyViz is very helpful with its example plugins [24].

TOSSIM provides runtime configurable debugging output, allowing a user to examine the execution of an application from different perspectives without needing to recompile. TinyViz is the java based graphical user interface that allows visualizing and controlling the simulation as it runs, inspecting debug messages, radio and UART packets, and so forth. The simulation provides several mechanisms for interacting with the network, packet traffic can be monitored, and packets can be statically or dynamically injected into the network.

TOSSIM is compiled by typing "make pc" in an application directory. In addition to the expected TinyOS components, a few simulator specific files are compiled. These files provide functionality such as support for network monitoring over TCP sockets. The TOSSIM executable is build/pc/main.exe. TOSSIM has a single required parameter; it is the number of nodes to simulate. To run a simulation of a single node "build/pc/main.exe 1" command should be used [24].

TOSSIM prints out all debugging information by default. TOSSIM debug output can be configured by setting the "DBG" environment variable in a shell. For example to see only the "crypto" and "led" type messages "export DBG=crypto,led" command should be used. This makes only LED and CRYPTO debug messages output enabled.

Most of the simulation and visualization scripts work by using these debug messages. Four DBG modes are reserved for application components and debugging use which are usr1, usr2, usr3, and temp. The debug message function has the following format:

dbg(<mode>, const char* format, ...);

The mode parameter in the function prototype specifies DBG modes under which this message will be printed. The full set of modes is defined in tos/types/dbg_modes.h file. The format and other optional parameters specify the string to output and have printf() semantics. Multiple modes can be enabled when running the simulator and a single debug message can be activated on multiple modes by using standard logical operators [24]. For example for the following function call the debug messages can be seen if either temp or usr1 is enabled.

dbg(DBG_TEMP|DBG_USR1, "Counter: Value is %i\n", (int)state);

Another advantage of TOSSIM is the usage of traditional debugging tools such as "gdb" as the application runs natively on a PC. However, it is stated that because TOSSIM is a discrete event simulation for large numbers of motes, traditional step through debugging techniques only work on an event basis, and not cross events.

"*TOSSIM replaces some of the TinyOS components, the components that handle interrupts and the main component. Interrupts are modeled as simulator discrete events. Normally, the core TinyOS loop that runs on motes is:*

```
while (1){
    TOSH_run_task();
}
```

TOSH_run_task runs tasks until the task queue is empty, at which point it puts the CPU to sleep. An interrupt will wake the mote. If the interrupt has caused a task to be scheduled, then that task will be run. While that task is running, interrupts can be handled. The core TOSSIM loop is slightly different:

```
for (;;) {
    while(TOSH_run_next_task()) {}
    if (!queue_is_empty(&(tos_state.queue))) {
        tos_state.tos_time = queue_peek_event_time(&(tos_state.queue));
        queue_handle_next_event(&(tos_state.queue));
    }
}
```

A notion of virtual time (stored as a 64-bit integer) is kept in the simulator (stored in tos_state.tos_time), and every event is associated with a specific mote. Most events are emulations of hardware interrupts" [24].

The nesC compiler modifies code generation for components such that fields declared in components result in arrays of each field when compiling for TOSSIM. The maximum number of motes that can be simulated at once is set at compile time by the size of this array. The default value is 1000 and is specified in /apps/Makerules with the "-fnesc-tossim-tosnodes" flag.

Energy is a crucial constraint in wireless sensor networks. Because of the cost and difficulty of deploying a sensor network, it is imperative to be able to obtain the power profile of an application before deployment. There is a tool called PowerTOSSIM which is a power modeling extension to TOSSIM. PowerTOSSIM accurately models power consumed by TinyOS applications [25].

PowerTOSSIM keeps track of state transitions of the various mote components and uses these in conjunction with a pluggable power model to compute the energy usage for each component of each simulated mote. The provided power model was derived from extensive measurements of the Mica2 mote, and it is validated that it yields extremely accurate results for a range of applications.

11.3 Comparison and Results

For comparing TinySec authentication-only feature with RC5 and Skipjack, TinySec internal modules are modified. TinySec internal modules are located at tos\lib\TinySec directory. TinySecC.nc, TinySecAppC.nc and crypto.h files are changed in order to use the TinySec with RC5, because current implementation is designed to use Skipjack.

The TinySec stack is changed to use the RC5M.nc module. The RC5 module contains three main functions for encryption, decryption and for the key setup phase. The cipher component in TinySecC.nc is switched to use RC5 and Skipjack to compare two algorithms. SkipJackM and RC5M modules are used for this purpose. The line for adding the Skipjack component “SkipJackM as Cipher” is changed to “RC5M as Cipher”.

The module TinySecAppC.nc is also changed for correct block cipher operations. CBCModeM.BlockCipher is changed to use RC5M.BlockCipher and CBCModeM.BlockCipherInfo is changed to use the RC5M.BlockCipherInfo components. The line “CBCModeM.BlockCipher -> SkipJackM.BlockCipher” is changed to “CBCModeM.BlockCipher -> RC5M.BlockCipher” and the line “CBCModeM.BlockCipherInfo -> SkipJackM.BlockCipherInfo” is changed to “CBCModeM.BlockCipherInfo -> RC5M.BlockCipherInfo”.

A sample test application is also developed for performance comparisons. For this purpose TestTinySec sample application is modified. Packet sending times and power consumptions are calculated. For sending and receiving messages two interfaces are used :

```
    interface SendMsg as Send;
    interface ReceiveMsg as ReceiveIntMsg;
```

GenericComm component's send and receive operations are bound to these interfaces from the configuration file. The sample configuration file is as follows:

```
includes IntMsg;
configuration TestTinySec {
}
implementation {
    components Main,
    GenericComm as Comm,
    TestTinySecM,
    Counter,
    LedsC,
    TimerC,
    TinySecC;
  Main.StdControl -> TestTinySecM.StdControl;
  Main.StdControl -> Comm.Control;
  Main.StdControl -> Counter.StdControl;
  Main.StdControl -> TimerC.StdControl;
  TestTinySecM.Send -> Comm.SendMsg[AM_INTMSG];
  TestTinySecM.ReceiveIntMsg -> Comm.ReceiveMsg[AM_INTMSG];
  TestTinySecM.TinySecMode -> TinySecC.TinySecMode;
  TestTinySecM.Leds -> LedsC;
  Counter.Timer -> TimerC.Timer[unique("Timer")];
```

```
  Counter.IntOutput -> TestTinySecM.IntOutput;
}
```

By using this configuration file the implementation of the module file can be coded. For example Comm.SendMsg is bound to interface TestTinySecM.Send and when a TinySec message sending is done the “sendDone” event is called with the appropriate parameters. The application is triggered by that event after message sending completed. In the below code the leds are set to green when the message sending completed.

```
event result_t Send.sendDone(TOS_MsgPtr m, result_t s) {
    call Leds.greenToggle();
    return s;
  }
```

When a new message is received at the application layer another event is called. This “receive” event is called with a pointer parameter that holds the incoming message. By using this message structure, parameters like message length and message address can be fetched. In the below code the leds are set to red when a new message is received. Also a debug message is output.

```
  event TOS_MsgPtr ReceiveIntMsg.receive(TOS_MsgPtr m) {
    call Leds.redToggle();
    dbg(DBG_USR1,"Msg received application layer.\n");
    return m;
  }
```

For sending data from the application with TinySec enabled, the data transmit mode for TinySec should be selected. This is done by calling the “TinySecMode.setTransmitMode” routine. For using TinySec Auth the following line will be used:

```
     call TinySecMode.setTransmitMode(TINYSEC_AUTH_ONLY);
```

For using TinySec AE the following line will be used:

```
    call TinySecMode.setTransmitMode(TINYSEC_ENCRYPT_AND_AUTH);
```

Sending the data from the application is done by calling “Send.send” routine. The following sample task broadcasts a 8 byte data to the network:

```
 task void sendit(){
    struct TOS_Msg data;
    (data.data)[0] = value;
    memset(data.data+1,0,7);
    (data.data)[7] = 0xff;
    call Send.send(TOS_BCAST_ADDR,8,&data);
 }
```

TOSSIM and PowerTOSSIM are used for calculating time and power consumptions. TinyViz is also used for debugging and visualization operations. The application is simulated for 600 seconds and calculations are done for 600 seconds. For building the application and generating time and power consumptions messages the following commands are used:

- export DBG=power,usr1
- export SIMDBG=power,usr1
- make pc
- build/pc/main.exe -t=600 -p 1 > apptrace.txt
- postprocess.py --sb=0 --em mica2_energy_model.txt apptrace.txt

The “-t” parameter given to “build/pc/main.exe” specifies the simulation running time and the “-p” parameters enables power profile usage. “postprocess.py” is the PowerTOSSIM tool’s script to calculate energy consumption. The PowerTOSSIM tool’s all needed scripts and power model files can be found in $TOSROOT/tools/scripts/PowerTOSSIM directory [25].

For testing the TinySec-Auth with Skipjack and RC5 the same application is compiled twice after changing TinySec library. As it is seen on the following table

(Table 11.1), the simulation is run for 600 seconds. One mote sends packets continuously with a timer.

Table 11.1: RC5 and Skipjack comparison with sample application.

	Skipjack	RC5
Simulation Time (s)	600	600
Number of Packets Sent	2423	2449
CPU Total (10^{-3}Joule)	7329.107964	7407.753179
Radio Total (10^{-3}Joule)	12605.250224	12740.511240
Total Energy (10^{-3}Joule)	19934.358188	20148.264419

It can easily be seen that RC5 is slightly faster than the Skipjack implementation. By using the RC5 implementation 2449 packets can be sent in 600 seconds and by using the Skipjack implementation only 2423 packets can be sent.

According to the output of the PowerTOSSIM tool the required energy values to send a packet with RC5 or Skipjack are nearly the same (Table 11.2).

Table 11.2: RC5 and Skipjack per packet energy consumption.

	Skipjack	RC5
CPU per packet (10^{-3}Joule)	3.0248072488	3.0248073413
Radio per packet (10^{-3}Joule)	5.2023319125	5.2023320702
Total per packet (10^{-3}Joule)	8.2271391613	8.2271394115

PowerTOSSIM tool can also generate a table containing time and current information. The time and current diagram for the RC5 and Skipjack implementation on the following figures shows that the values are between 11.16 and 12.6 mA (Figure 11.4 and 11.5).

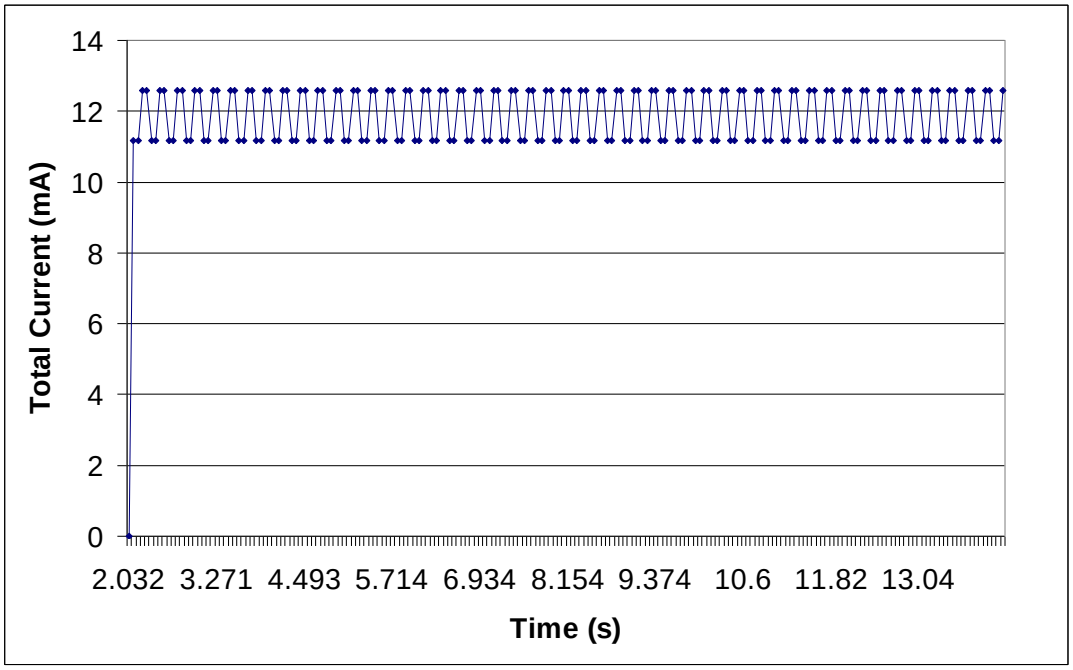


Figure 11.4: Total current, plotted as a function of time (RC5).

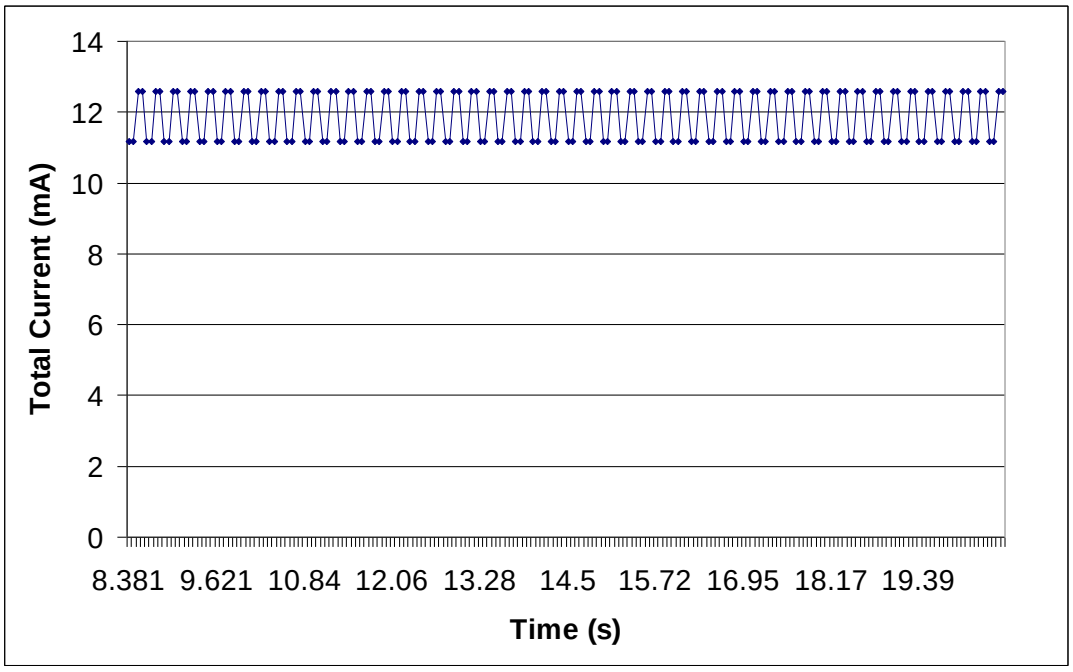


Figure 11.5: Total current, plotted as a function of time (Skipjack).

By using the TOSSIM simulator, the time needed for a packet to be sent is calculated and it seems that using RC5 for authentication is slightly faster. For future work, these tests may also be done for a network with a lot of nodes with variable node density. Mobile nodes and some outsider attacks can also be considered for bandwidth and latency calculations.

12. CONCLUSIONS

The security issues for sensor networks are different than the ones for fixed networks. The security requirements are availability, confidentiality, integrity, authentication, and non-repudiation, but they are considered differently for sensor networks.

This is due to system constraints in mobile devices and frequent topology changes in the network. System constraints include low-power microprocessor, small memory and bandwidth, and limited battery power. In addition to usual denial of service attacks, availability in sensor networks can be threatened by radio jamming and battery exhaustion [9].

Also there are many security issues in sensor networks this work focused on authentication. This is the core requirement for integrity, confidentiality and non repudiation.

TinySec is a good implementation of an authentication and encryption algorithm where there are significant resource limitations like energy and computation power. TinySec uses the security primitives that have been investigated and used by the security community in many years. There are also some researchers building key exchange protocols for TinySec as it is completed and ready in TinyOS source code.

The energy consumption increase of TinySec is 10% with authentication and encryption is used which is most resource using and most secure mode. On the other hand it causes an increase of 3% energy consumption for TinySec-Auth which only does authentication using MACs. The algorithm also proved that software based link layer security is applicable for resource constrained devices because it has low effects on bandwidth and latency.

In this work, TinySec authentication-only feature is tested with RC5 and Skipjack algorithms for computing the message authentication codes. For this purpose TinySec internal modules are modified. TOSSIM and PowerTOSSIM are used for calculating time and power consumptions. It is seen that RC5 has a longer key setup time but its encryption and decryption routines are faster than Skipjack's.

REFERENCES

[1] **Jubin, J. and Tornow, J.D.**, 1987. The DARPRA Packet Radio Network Protocol, *Proc. IEEE,* **75,** 1.

[2] **Akyıldız, I.F., Su, W., Sankarasubramaniam, Y. and Çayırcı, E.**, 2002. Wireless sensor networks: a survey, *Computer Networks,* **38,** 393-422.

[3] **Wang, A., Heinzelman, W. and Chandrakasancar, A.**, 2000. Energy-Scalable Protocols for Battery-Operated Microsensor Networks, *Proceedings of the ARL Federated Laboratory 4th Annual Symposium,* College Park, March 21-23.

[4] **Carman, D.W., Kruus, P.S. and Matt, B.J.**, 2000. Constraints and Approaches for Distributed Sensor Network Security, Glenwood, MD.

[5] **Mills, D.,** 2000. Low Energy Communications and Routing for Microsensor Networks, *Proceedings of the ARL Federated Laboratory 4th Annual Symposium,* College Park, March 21-23.

[6] **Wood, A. and Stankovic, J.**, 2002. Denial of service in sensor networks, *IEEE Computer*, **35,** 54-62.

[7] **Aether Wire Location, Corp.,** 1995. Low-Power, Miniature, Distributed Position Location and Communication Devices Using Ultra-Wideband, Nonsinusoidal Communication Technology, *Semi-Annual Technical Report,* **J-FBI-94-058,** Aether Wire Location Corp..

[8] **Abdul-Rahman, A. and Hailes, S.,** 1997. A Distributed Trust Model, *New Security Paradigms Workshop 1997,* ACM.

[9] **Weimerskirch, A., Thonet, G.,** 2001. A Distributed Light Weight Authentication Model for Ad-hoc Networks, *The 4th International Conference on Information Security and Cryptology (ICISC 2001),* December 06-07, 341-354.

[10] **Stajano, F. and Anderson, R.,** 1999. The Resurrecting Duckling: Security Issues for Ad hoc Wireless Networks, *The 7th International Workshop on Security Protocols*, LNCS 1796, Springer-Verlag.

[11] **Gennaro, R., Jarecki, S., Krawczyk, H. and Rabin T.,** 1996. Robust Threshold DSS Signatures, *Advances in Cryptology*, LNCS 1070, Springer-Verlag.

[12] **Boghe, M., Trappe, W.,** 2003. An Authentication Framework for Hierarchical Ad Hoc Sensor Networks, *Proceedings of the ACM Workshop on Wireless Security,* Westin Peachtree Plaza, Atlanta, Georgia, U.S.A, September 28.

[13] **Perrig, A., Canetti, R., Song, D., and Tygar, D.**, 2001. Efficient and Secure Source Authentication For Multicast, *Proceedings of the Network and Distributed System Security Symposium (NDSS2001)*, 35-46.

[14] **Zhu, S., Xu, S., Setia, S., Jajodia, S.**, 2003. LHAP: A Lightweight Hop by Hop Authentication Protocol For Ad-Hoc Networks, *International Workshop on Mobile and Wireless Network (MWN 2003),* Providence, Rhode Island, USA, May 19-22.

[15] **Zhu, S., Xu, S., Setia, S., Jajodia, S.**, 2003. LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks, *Proceedings of the 10th ACM Conference on Computer and Communications Security*, Washington D.C., October.

[16] **Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V. and Culler, D.E.**, 2002. SPINS: Security Protocols for Sensor Networks, *Kluwer Wireless Networks*, **8**, 521-534.

[17] **Venkatraman, L., and Agrawal, P.**, 2000. A Novel Authentication Scheme for Ad hoc Networks, *Wireless Communications and Networking Conference (WCNC 2000),* 1268-1273.

[18] **Slijepcevic, S., Potkonjak, M., Tsiatsis, V., Zimbeck, S. and Srivastava, M.B.**, 2002. On Communication Security in Wireless Ad-Hoc Sensor Networks, *WETICE 2002*.

[19] **Karlof, C., Sastry, N. and Wagner, D.**, 2004. A Link Layer Security Architecture for Wireless Sensor Networks, *Proceedings of the*

Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004), November.

[20] **Li, T., Wu, H., Bao, F., Lee, H.,** 2004. TinySec* Security API of TinyOS, *http://www.i2r.a-star.edu.sg/icsd/SecureSensor/papers,* July 1.

[21] **National Security Agency,** 1998. SKIPJACK and KEA Algorithm Specifications, *http://csrc.nist.gov/CryptoToolkit/skipjack/skipjack-kea.htm,* May 29.

[22] **Rivest, R.L.**, 1994. The RC5 Encryption Algorithm, *Proceedings of the 1994 Leuven Workshop on Fast Software Encryption (Springer 1995),* November, 86-96.

[23] **Yin Y.L., Kaliski B.S.**, 1998. On the Security of The RC5 Encryption Algorithm, *RSA Laboratories Technical Report TR-602,* September.

[24] **Levis P., Lee N., Welsh M. and Culler D.**, 2003. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications, *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003),* September.

[25] **Shnayder V., Hempstead M., Chen B.R., Allen G.W. and Welsh M.**, 2004. Simulating the Power Consumption of Large Scale Sensor Network Applications, *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04),* Baltimore, MD, November.

CURRICULUM VITAE

Mehmet Erhan Yiğitbaşı was born in 1980. He received his Bsc degree from Istanbul Technical University Computer Engineering Department, in 2002 with highest GPA. He is currently working as a Software Engineer for a company in telecommunications field.