DEFENSE AGAINST INSIDER ATTACKS (SYBIL ATTACK) USING CODE TESTING

MCS

by

Imran Makhdoom

A thesis submitted to the faculty of Information Security Department Military College of Signals, National University of Sciences and Technology Rawalpindi in partial fulfillment of the requirements for the degree of MS in Information Security

APRIL 2015

SUPERVISOR CERTIFICATE

It is to certify that Final Copy of Thesis has been evaluated by me, found as per specified format and error free.

Dated___________ _________________________

(Dr Mehreen Afzal)

ABSTRACT

Wireless Sensor Networks (WSN) due to their distributed nature are vulnerable to various external and insider attacks. Classic cryptographic measures do protect against external attacks to some extent but they fail to defend against insider attacks involving node compromise. A compromised node can be used to launch various attacksof which Sybil Attack is the most prominent. Existing security protocols in WSN fail to provide protection against all the dimensions of Sybil Attack. Code attestation is considered to be the only potent defense against node compromise and related integrity attacks including Sybil Attack launched by change in the code of the end device. Various code attestation protocols do exist but they are either vulnerable to network attacks being challenge-response based or they are in-efficient with respect to performance and security aspects. One Way Memory Attestation Protocol (OMAP) is one of them. OMAP claims 90% detection rate in case 20% of the end device's memory is modified, but with increased time overhead.

A detailed review and analysis of various defenses proposed against Sybil Attack has been carried out. Their strengths and weaknesses have been identified and ultimately a novel One Way Code Attestation Protocol (OWCAP) for wireless sensors networks is proposed, which is an economical and a secure code attestation scheme that protects not only against Sybil Attack but also against majority of the insider attacks involving node compromise.It detects the modified memory of an end device with 100% detection rate when only 0.8 % memory is changed. The performance analysis of the proposed scheme OWCAP is carried out in detail by implementation in Dev C++ and Code Composer.

DEDICATION

All praise and thanks to almighty Allah, the most Gracious and the most Compassionate, Master of the Day of Judgment.

I dedicate my work to my parents, my wife and my teachers who have been a constant source of encouragement, guidance and payers for me, throughout my work.

ACKNOWLEDGEMENT

First of all, I would like to thank to Almighty Allah, who has blessed me with intellect and strength, Who increased my knowledge and made me able to successfully go through this phase and complete my work.

The first and the foremost person who I would like to acknowledge is my supervisor Dr. Mehreen Afzal. Despite of her busy schedule, she was always available for discussions and queries. She continuously kept me in the right direction by giving valuable suggestions. She used to ask for my progress time to time and after understanding my advancements, gave me appreciated ideas and recommendations. Without her constant support, help and encouragement, timely completion of my research work would have been impossible.

I am also immensely thankful to my guidance committee members Lt Col Dr. Imran Rashid, Lt Col Dr. Monis Akhlaq and Lec Mian Muhammad Waseem for their persistent support and inspiration.

I would also like to thank Dongwon Seo, one of the authors of OMAP, who helpedme in understanding OMAP, answered all my queries and encouraged me for the development of OWCAP.

I am also thankful to my Head of Department Lt Col Dr. Babar Aslam and his team for administrative help and support.

I am immeasurably thankful to my family members who took care of me throughout my MS studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Number *Page*

KEY TO ABBREVIATIONS

WSN	Wireless Sensors Network
OWCAP	One Way Code Attestation Protocol
OMAP	One Way Memory Attestation Protocol
ICS	Industrial Control System
PLC	Programmable Logic Controller
DoS	Denial of Service
XOR	Exclusive OR Operation
TOCTTOU	Time of Check to Time of Use
BS	Base Station
PKI	Public Key Infrastructure
SCADA	Supervisory Control and Data Acquisition
SWATT	Software based Attestations
BTSS	Basic Threshold Secret Sharing Scheme
MVBS	Majority Voting Based Scheme
ISA	Instruction Set Architecture
PRG	Pseudo Random Generator
MPU	Micro Processor Unit
MCU	Micro Controller Unit
SN	Serial Number
AMI	Advanced Metering Infrastructure
BIOS	Basic Input Output System

DRM	Digital Rights Management
MAC	Message Authentication Code
HMAC	Hash Message Authentication Code
ACK	Acknowledgement
SHA	Secure Hash Algorithm
ROP	Return Oriented Programming
BW	Bandwidth

C h a p t e r 1

Introduction

1.1 Overview

The Stuxnet worm that hit the industrial world with a bang had specifically targeted the Programmable Logic Controllers (PLCs) of Supervisory Control and Data Acquisition (SCADA) system and modified the executable code of PLCs so that they are deviated from their expected behavior [1], [2]. Stuxnet was developed to be a self-directed smart code that could independently communicate, update, target and spread. It had the capability to impersonate the normal behavior of PLC by stealthily surviving in a large scale system. It faked the PLC readings until a physical damage to the Industrial Control System was done. Any Network Management System (NMS) was failed to identify any malfunction in the routine operation of PLCs[3].In similarity to other worms, Stuxnet spread erratically in the computer network, but the quality that makes it different from its predecessors is its capability to unleash its payload at a PLC of a specific Industrial Control System (ICS) that matched the characteristics of ICS installed at Iran's nuclear enrichment facility at Natanz. Subsequently Stuxnet altered the codeof the PLC that controlled the thousands of centrifuges, thus ultimately disrupting the Uranium enrichment by damaging almost all the centrifuges [4][5].The successful malicious execution of Stuxnet motivated various other attackers to develop different software based attacks like Boden, Flame and Doqu that also targeted ICS [6].Stuxnet is not the end, there is a continuous effort put in by the attackers to acquire requisite knowledge, skills and tools to attack different components or segments of ICS[7].Wireless Sensors Network is also one of them.

The wireless sensors are low cost, low power, self organizing and easy to deploy which makes them an ideal candidate for distributed monitoring of a prescribed area. However the distributed nature of deployment on one side is beneficial but on the other side due to the lack of physical protection they are prone to external and insider attacks. The insider attacks launched as a result of physical compromise of a sensor node are the hardest to protect against. In external attack, an attacker can easily intercept, modify, replay or inject fabricated messages [8]. Different cryptographic schemes have been proposed to protect against threats to confidentiality and authentication but the insider attacks are not preventable with only classic cryptographic techniques [9][10][11] and [13]. The worst aspect of node compromise is that an adversary can install malicious program in the compromised node and can play havoc with the legitimate functioning of the WSN by launching various attacks including, selective forwarding attack, worm hole attack, hello flood attack [14], DoS attack [15] and above all a multi-dimensional Sybil Attack [16].

WSN has a constrained environment in terms of limited memory, storage and processor capabilities along with stern energy limitations [17]. These constraints make it very challenging to design new security protocols to protect against Sybil Attack, launched as a result of a node compromise. The key idea here is that the code running on a compromised node is different than the legitimate node and it is the malicious code which enables the particular node to launch the Sybil Attack or other attacks mentioned earlier. Various defense techniques and protocols [16], including SPINS [36], TinySec[39], MiniSec [45], ZigBee [46] and LEAP+ [47]have already been developed and implemented in Wireless Sensors Networks but they all have some residual weaknesses. Either they do not protect against all the dimensions of Sybil Attack or they fail to identify a compromised node.

Code Attestation is the only technique that is considered to be a potent defense not only against Sybil Attack [16] but also provides implicit protection against other insider attacks that involve a change of code as a result of a node compromise. Different types of Code Attestation Protocols have already been proposed [22], [23] and [24], but they have some blatant weaknesses. The major problems in all the existing Code Attestation Schemes are, difference in Time of Check to Time of Use (TOCTTOU) [25], elusive start of code test procedure, and independence of code test procedure to routine application running on a wireless sensor node. The difference in TOCTTOU can render protection provided by any code attestation scheme ineffective, in case attacker exploits this time gap and make any modification in the code just before the wireless sensor is suppose to send its scheduled updates to base station or sink node. The independent code attestation procedures result in excessive bit transmission over and above bits transmitted for sensor reading updates. This excessive bit transmission exhausts the available energy on a sensor node thus reduces its lifetime.

1.2 Need forResearch

The use of wireless Sensor Networks is becoming ubiquitous in battle field surveillance, ICS / process controls / automation, traffic management systems, environmental monitoring, smart grids, nuclear power plants, smart houses, pipe line monitoring and etc. But with the advent of attacks like Stuxnet, the integrity of information travelling between PLCs and the sensors is of utmost importance. A slight change in the code of a sensor mote or a PLC can now cause an irreparable physical damage to the ICS. The integrity of information is of vital importance in case the sensors are employed in a sensitive control systems environment like nuclear power plants, gas pipe line monitoring or battle field surveillance. Due to the distributed nature of deployment, wireless sensors are prone to physical compromise and as a result multiple insider attacks can be launched by altering the code of the sensor motes. Code attestation is considered to be the most effective defense against insider attacks, if efficiently performed it can identify a malicious or altered code with 100 percent accuracy. At present there are various code attestation schemes already proposed / published but they are either vulnerable to network attacks or fail to abstain an attacker to violate the integrity of messages passed over wireless sensors networks. Therefore there is a need to develop an efficient and secure code attestation scheme.

1.3 Problem Statement

The existing Code Attestation Protocols [22], [23] are challenge response based so they are prone to network attacks like rainbow attack, interference attack and other Man-in-the-Middle (MITM) attacks. The protocol mentioned in [24] is although a One Way Memory Attestation Protocol (OMAP) but in addition to its specific weaknesses the common weakness that it shares with its predecessors is the difference in TOCTTOU. The gap in TOCTTOU always put a question mark on the integrity of data / message being transmitted over the wireless sensors network. OMAP has its other specific weaknesses that include, poor time synchronization, more computation time, limited memory address traversal, and the most critical is low detection rate of malicious code. Moreover the independent code attestation procedures result in excessive bit transmission over and above bits transmitted for sensor reading updates. This excessive bit transmission exhausts the available energy on a sensor node thus reduces its lifetime. In order to address the issues of security, transmission overhead and energy consumption there is a need to develop an efficient and a secure code attestation protocol.

1.4 Objectives

1. *To Carryout an analysis of existing Code/Memory Attestation Protocols.*

2. *Developing an efficient and a secure Code Attestation Protocol for WSN with implicit Data Authentication.*

3. *Carryout the performance analysis of the proposed scheme w.r.t computation time, detection rate, and calculating time interval of sensor response update depending upon available energy.*

1.5 Research Methodology and Achieved Goals

The research work has been divided into three main phases. In the first phase, detailed study and literature review has been carried out related to the Sybil Attack. A strong theoretical concept has been built regarding the dimensions of Sybil Attack, insider attacks and defense against such attacks using Code Attestation. In the second phase, the development of a Code Attestation Algorithm using various specifications has been investigated. Dev C++has been used for the initial testing because it was found suitable for the initial development of the project. The proposed OWCAP Algorithmhas been implemented in Dev C++ for the comparative analysis of multiple variants of this scheme. After thorough testing, the code has been implemented in Code Composer to analyze the performance of the code in the microcontroller environment.

1.6 Thesis Organization

The thesis report has been divided into 6 chapters. In Chapter 2various defenses proposed against Sybil Attack are discussed in detail. Chapter 3 presents a critical analysis of some popular code attestation schemes. Chapter 4 introduces an economical and a secure One-Way Code Attestation Protocol (OWCAP). Chapter 5 carries out a unique comparative analysis of various code attestation schemes and Chapter 6concludes the thesis report by giving the scope of future work.

C h a p t e r   2

Literature Review

2.1 Introduction

In this chapter an overview of threats to WSN is given along with various known defenses proposed against Sybil Attack are reviewed.These defense techniques are evaluated based upon their ability to defend against all the dimensions of Sybil Attack and also to protect against a Sybil Attack launched from a compromised node with malicious code modification.

2.2 Threats to WSN

Due to the distributed nature of deployment of sensor nodes in WSN many attacks are possible. These attacks fall under one of the following categories [15], attacks on secrecy and authentication, attacks on network availability and stealthy attack against service integrity.Based upon attacker’s access to the WSN the attacks can be classified as external and internal (insider) attacks. The external attacks are launched by an attacker, who does not belong to the network and the internal attacks are launched by an adversary who has the access of the network or they know the cryptographic keys, thus they form part of the network and then launch various attacks. Insider attacks that are launched from a compromised node are much harder to detect. Few examples of external and internal / insider attacks are:-

2.2.1 Eavesdropping

Due to broadcast nature of wireless communication in WSN, an attacker can easily sniff the traffic of the WSN.

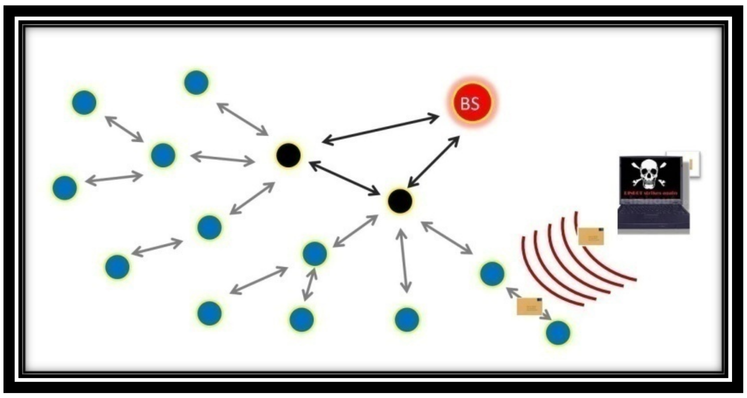


Figure 2.1 Eavesdropping

2.2.2 Replay

A message transmitted by a legitimate node at a time T0 is retransmitted / replayed at a later time T4 by a malicious node.

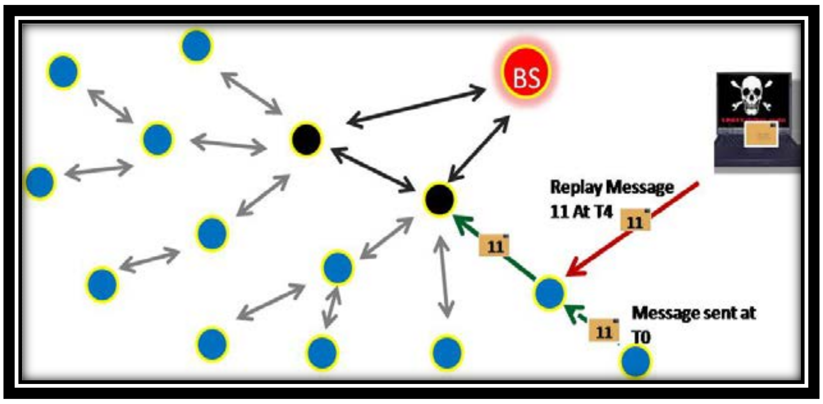


Figure 2.2 Replay Attack

2.2.3 Fabricated Message

A malicious node injects false messages in the network, hence attacking the authentication of the messages being relayed in the network.

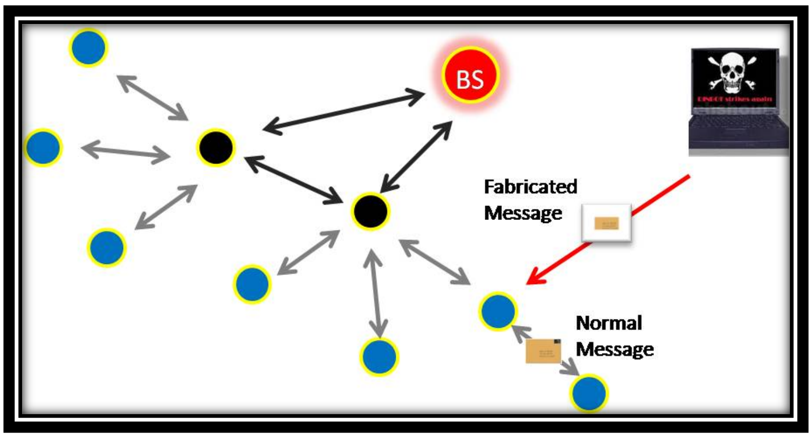


Figure 2.3 Fabricated Messages

2.2.4 Interruption

It is one of the DoS attacks, which makes the network unavailable by causing interruption to transmission of messages. Such an interruption can be achieved by jamming, collision or misdirection.

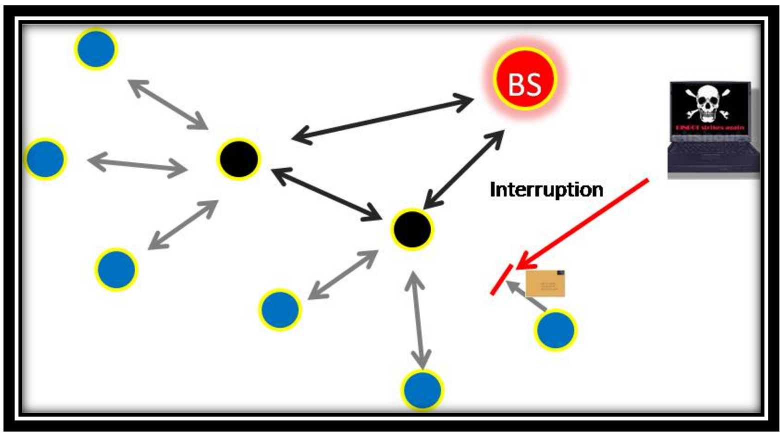


Figure 2.4 Interruption

2.2.5 Selective Forwarding Attack

In this attack a malicious node (made after compromising a legitimate node) routes packets from selective nodes only and drop packets from others thus rendering affected nodes to DoS.

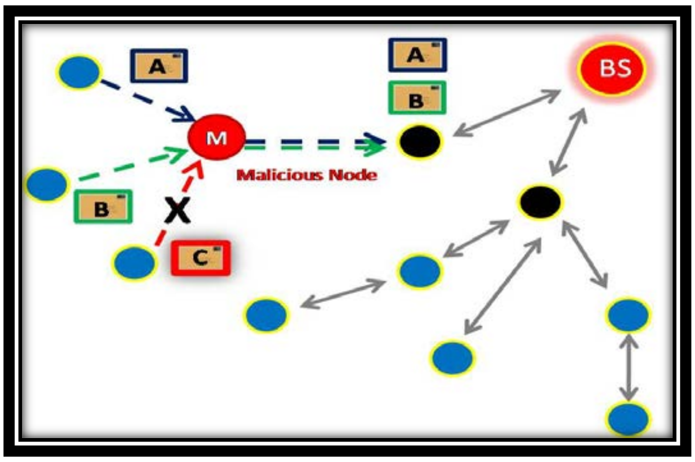


Figure 2.5 Selective Forwarding

2.2.6 Worm Hole Attack

In this attack a malicious node receives messages at one pt in the network and tunnels them at another point, thus drastically affecting the routing protocols. The attack can be launched in-spite of the various network security measures to include confidentiality and authenticity and also without any information about cryptographic keys.

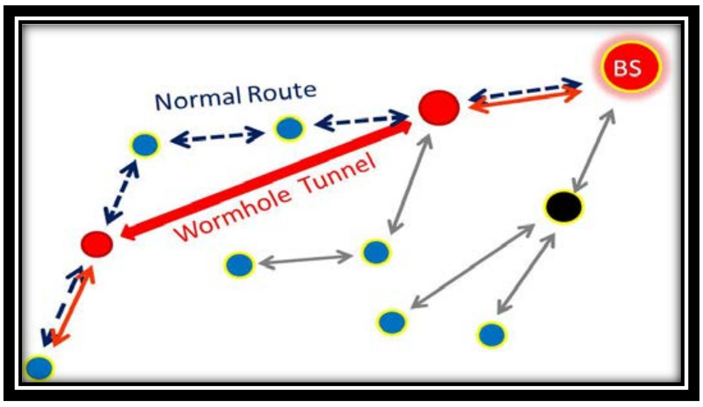


Figure 2.6 Worm Hole Attack

2.2.7 Hello Flood Attack

In order to disrupt the routing protocol and network topology a laptop class attacker, broadcasts “Hello” messages with high power to distant nodes, so that the nodes once receive hello message consider the malicious node as their immediate node and try to route messages to BS via malicious node. This will not only affect the routing table of a particular node but will also drain the precious battery power of the node by continuously forwarding messages to the malicious node and not receiving any acknowledgement.

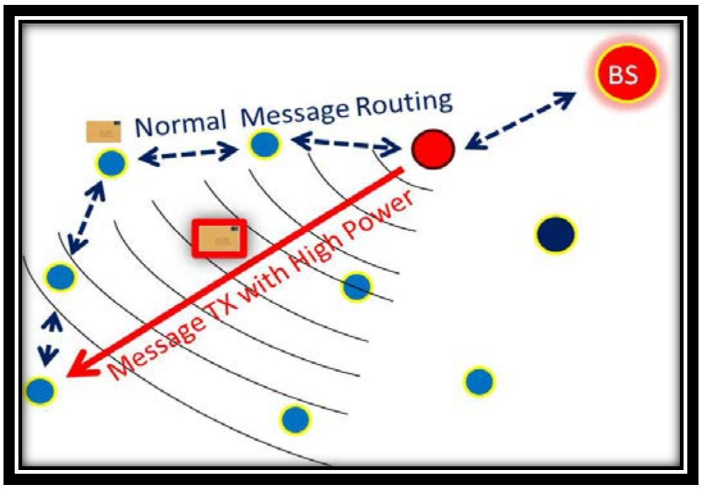


Figure 2.7 Hello Flood Attack

2.2.8Sybil Attack

In this attack a malicious node illegitimately takes on multiple identities. The additional identities are called the Sybil Nodes. The malicious node can fabricate or steal the identities of the legitimate nodes. Sybil Attack adversely affects the normal functioning of the sensor network such as distributed storage, dispersion and multipath routing.

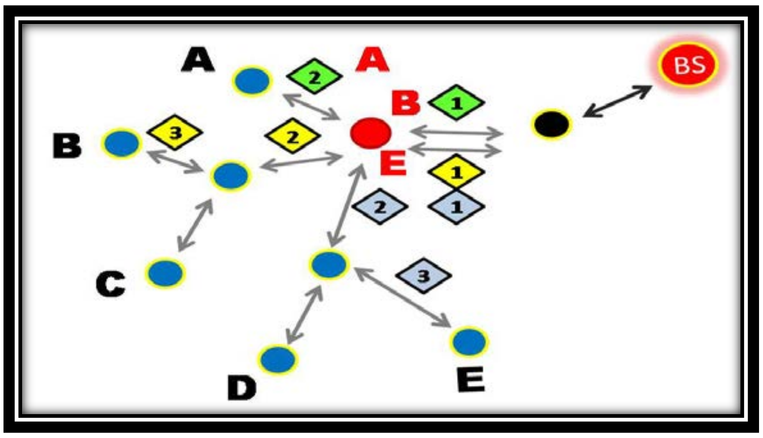


Figure 2.8 Sybil Attack

2.3 Taxonomy of Sybil Attack

This section discuss the Sybil Attack in detail with some brief history that it is an attack in which a malicious node/entity illicitly take multiple pseudonyms to deceive others, thus harnessing detrimental effects on computer and network security. It was initially termed as “**Pseudo Spoofing”** before 2002 but later it was named “**Sybil**” after the subject of a book, which was about a woman with “Multiple Personality Disorder” i.e one person under the control of two distinct identities[48]. A malicious node pretendsto be multiple nodes at a same time using a single physical device. Attacker can have multiple identities by stealing identities of other nodes, forging false identities, generatingrandomnumberofnew identities, stealing or modifying cryptographic primitives by physically compromising the sensor nodes and physically compromising sensor nodes, and installing malicious code.Sybil Attack in Wireless Sensor Networks poses a potent threat to, data aggregation, routing mechanism, distributed storage and fair resource allocation.The attack platform can be static or mobile. In case the attacker launching the Sybil Attack is static and the key point here is that illegitimate multiple identities are generated from the same location. Therefore any protection scheme that incorporates the physical location of each node, can easily detect that multiple identities are being generated from the same location. Contrary if the attacker launching the Sybil Attack is mobile and presents multiple identities from different locations then it is difficult to detect the Sybil Identities.An attacker launching a Sybil Attack may operate in one of the following dimensions[16]:-

2.3.1 Dimension-1 Direct or Indirect Communication

In direct communication, Sybil Nodes directly communicate with legitimate nodes. If a legitimate node sends a message to a Sybil Node, it is actually received by one of the malicious node. And in indirect communication, legitimate nodes cannot communicate directly with Sybil Nodes, rather one or more malicious nodes route messages to / from the Sybil Nodes.

2.3.2 Dimension-2 Fabricated or Stolen Identities

Attacker can generate random new identities but if there is an identity management mechanism in WSN network, then generating new identities may be difficult for the attacker. Now the only viable option left with attacker is to steal the identities of legitimate nodes. This can get even worse if the attacker physically compromise sensor motes.

2.3.3 Dimension-3 Simultaneous or Non-Simultaneous

The attacker tries to make all the Sybil Nodes appear simultaneously in the network. He can do so by cycling through these identities one by one.Attacker may resort to introducing multiple physical devices in the network. These multiple physical devices switch identities at different time intervals, a particular identity may only be used once.

2.4 Existing Security Protocols in WSN

Since 2000 various forms of cryptographic defense mechanisms against various attacks in WSN have been developed or proposed. Most of the security protocols have compromised on security in order to achieve efficiency in a constrained environment. None of these security suits address the issue of Sybil attack in totality. Some of these defense mechanisms are discussed in the following sections.

2.4.1 SPINS[36] a suite of security building blocks optimized for resource constrained environments and wireless communication was presented in 2001. SPINS has two secure building blocks: SNEP and µTESLA. SNEP provides data confidentiality, two-party data authentication, and weak data freshness. µTESLA is a new protocol which provides authenticated broadcast for severely resource-constrained environments. SNEP uses RC-5 CTR for encryption and µTESLA uses CBC-MAC for authentication. It is a conventional 2 pass encryption and authentication technique which provides only weak freshness. RC-5 is a simple cipher but does have weaknesses [37]. Distributed.net has brute-forced RC5 messages encrypted with 56-bit and 64-bit keys, and is working on cracking a 72-bit key; as of Dec 2012, 2.671 % of the key space has been searched [38]. SPINS does not provide protection against insider attacks including Sybil Attack[40].

2.4.2 TinySec[39] introduced in 2004 was implemented in TinyOS as a link layer security protocol. It provides authenticated encryption using Skipjack in CBC mode for encryption and CBC-MAC for authentication. However CBC-MAC is secure for fixed length messages only and not for variable length messages [44]. TinySec does not protect against replay attacks [40]. Eli Biham and Adi Shamir discovered an attack against 16 of the 32 rounds of Skipjack, within one day of declassification, and (with Alex Biryukov) extended this to 31 of the 32 rounds (but with an attack only slightly faster than exhaustive search) within months using impossible differential cryptanalysis. A truncated differential attack was also published against 28 rounds of Skipjack cipher.This was later complemented by a slide attack on all 32 rounds [41]. According to the claims of RSA Security Labs, 80-bit keys are not secure after 2010 [42][43]. In TinySec the MAC protects the entire packet, including the destination address, AM type (active message handler), length, source address and counter (if present), and the data (whether encrypted or not). It does not prevent insider attacks as a result of node compromise.

2.4.3 MiniSec[45] published in 2007. It is a network layer security protocol that provides authenticated encryption but does not provide exclusive protection against insider attacks including Sybil Attack.

2.4.4ZigBee[46]is a low-cost, low-power, wireless mesh network standard which was made publicly available in 2005. It usesAEAD_AES_128_CCM mode for authenticated encryption, the algorithm works as specified in [CCM], using AES-128 as the block cipher, by providing the key, nonce, associated data, and plaintext to that mode of operation. It does not provide exclusive protection against insider attacks as a result of node compromise.

2.4.5 LEAP+[47] introduced in 2006 is a key management protocol for sensor networks that is designed to support in-network processing, while at the same time restricting the security impact of a node compromise to the immediate network neighborhood of the compromised node. It uses RC-5 Cipher for encryption and CBC-MAC. It does provide data integrity but no protection against Sybil Attack.

2.5 Existing Defenses Against Sybil Attack

2.5.1 Radio Resource Testing

Radio Resource Testing is a form of resource testing with considerable weaknesses [16]. Firstly, the probability of detecting a Sybil Node is very low in case we have less number of channels than the neighboring nodes. Secondly, it does not protect against an attacker which physically compromise a node and modify its program to launch a Sybil Attack. Thirdly, it also does not protect against an attacker, who does not present all the Sybil Identities simultaneously.

2.5.2 Key Pool

Key Pool is another such scheme in which random keys are assigned to each node from a pool of X keys [16]. One of its weaknesses is that during validation sensor nodes consume a lot of energy. Secondly, during validation especially full validation, channel bandwidth is reduced for control and sensor messages. Thirdly, node to node authentication is not possible because during random distribution of keys, keys can be issued multiple times out of the key pool. Fourthly, if an attacker compromises sufficient nodes, he can use the stolen identities.

2.5.3 Multi-Space Pair wise Key Distribution

In Multi-Space Pairwise Key Distribution Scheme, the setup server randomly generates a pool of m key spaces and each sensor node is assigned p out of the m key spaces [16]. This scheme consumes precious memory space, as every node is required to store pair wise keys with its neighbors. Although the attacker cannot generate new IDs until a threshold of captured nodes is met, yet he can use the limited number of IDs he has captured from compromised nodes to send false sensor responses. The pair wise key distribution also does not protect against other integrity attacks, in which code of the compromised node has been tampered with.

2.5.4 Position Verification

Position verification is considered to be a potent defense against Sybil Attack. The solutions proposed consider that the network is static, thus verifying the positions of all the nodes and detecting Sybil Nodes that appear to be at the same physical location [19],[20] and [21]. The drastic weakness in these solutions is that, they do not protect against a mobile attacker. Location verification also does not protect against other integrity attacks, in which a compromised node has been tampered with. Moreover the positions of the end nodes are required to be unique for each identity.

2.5.5 Trusted Certification

Trusted Certification is another promising scheme that has the potential to protect against Sybil Attacks, provided there is a trusted central authority [12], [18].There is no method for ensuring uniqueness of certificates, and practically it has to be done by a manual setup. Whereas this manual setup can be costly, and also causes performance degradation. Moreover, the process of issuing and managing certificates is computation intensive. Related to trusted certification, we have another scheme called Asymmetric Encryption in which Public Key Infrastructure (PKI) is used as a defense against Sybil Attack. The key distribution may be easier as compared to symmetric keys but the length of the keys outweighs the memory resource available at each node.

2.5.6 Code Attestation

Code Attestation protects against all the dimensions of a Sybil Attack [16]. It also protects against almost all the attacks involving change of code. Although this scheme is considered to have high computation requirement compared to pairwise key distribution schemes yet many such schemes have been proposed [22],[23] and [24].

2.6 Summary

Sybil attack has multiple dimensions, thus development of a defensive scheme like code attestation that protects against all the dimensions of Sybil Attack has inherent advantage of protecting against all the other insider attacks involving node compromise. Any attack launched from a compromised node with change in program code can be detected by code attestation. Therefore code attestation is not only a suitable defense for embedded devices but also for all other con systems as well, that have distributed nature of sensor deployment like SCADA systems.

Chapter 3

CODE ATTESTATION

3.1 Introduction

Many different cryptographic schemes have been proposed to protect against threats to confidentiality and authentication but the inside attacks are not preventable with only the classic cryptographic techniques. These attacks mainly include node compromise which is another major problem for WSN security. The worst aspect of node compromise is that an adversary can install malicious program in the compromised nodes and can play havoc with the legitimate functioning of the WSN. The only efficient and economical protection against such an attack is code attestation.

Code attestation is of two types, hardware based attestation and software based attestation. An example of hardware based attestation is a Trusted Platform Module (TPM) developed by Trusted Computing Group (TCG) [28], which is the tamper-evident chip that performs the attestation and ensures that initial information stored by the manufacturer in the memory is not modified. But such an arrangement is not feasible for sensor motes since TPM cannot update its software very often which is one of the basic requirements of a sensor mote. Moreover limitations of processing power, memory, power and cost does not allow deployment of TPM on sensor motes[32]. Whereas, software based attestation does not require an additional hardware and performs the attestation over a device entirely in software [33].

3.2 Review and Analysis of Code Attestation Schemes

As discussed in previous section that there are both, hardware and software based code attestation schemes, but considering the cost effect in setting up a large scale WSN, software based schemes are the ideal solution that can be implemented not only on latest but also on legacy sensor devices. Therefore in this chapter the focus will only be on the software based schemes.

3.2.1 Software Based Attestation for Embedded Devices (SWATT)

SWATT (Software Based Attestation) is a challenge-response based code attestation scheme [22], implemented on 8 Bit Harvard architectureAtmel AT-Mega 163L microcontroller with 16 KB program memory and 1 KB of data memory. The protocol is based on certain assumptions:Verifier has a correct view of the end device’s hardware, i.e Clock Speed, ISA (Instruction Set Architecture), Memory architecture of the microcontroller on device and Size of device’s memories, attacker does not change the device’s hardware i.e processor. In SWATT the end device contains a memory content verification procedure at the time of deployment and the verifier remotely activates the verification procedure by sending a random challenge. SWATT was developed to accomplish following objectives: External verifier can detect with high probability, if a single byte of the memory deviates from expected value, verification is software based and there is no need of extra hardware, so SWATT can be used on legacy systems.SWATT verifies contents of program memory.

3.2.1.1 SWATT Algorithm

Verifier sends (**Random Seed** and **m**)to the sensor node

// m is the number of iterations or number of accesses to be made to the memory addresses.It is sent by the verifier to the attested device.

Value of “m” depends upon the size of the attested device’s memory and is calculated based upon Coupon Collector’s Problem, which states that if “s” is the size of the memory then the iterations or accesses to be made to the memory such that each memory location is accessed at least once. $m = (s \ln s)$

// Random Seed : is the input to the Pseudo Random Generator (PRG) RC4

// Output : 64 bit checksum of the device’s program memory.

```
For  i  =  1  to m
        {
        Do
        // construct address for memory read
                    $A_i = (RC4_i << 8) + c_{[(j-1) \bmod 8]}$
                    ↗            ↑              ↖
     16 bit value of       8 x bits of RC4 Output     8 x bits of current checksum
memory address
         // Update Checksum Byte
$C_j = C_j + (\text{mem}[A_j] \oplus C_{(j-2)\bmod 8} + RC4$
$C_j$ = rotate left 1 bit
  // Update checksum index
j = (j+1) mod 8
return C
```

3.2.1.2 Pros and Cons of SWATT

The strong points of SWATT include:The use of random challenge avoidspre-computation and replay attack, memory addresses depend on the output of PRG, so the attacker does notknow beforehand as to which memory address will be verified, each byte of checksum (each iteration of for loop) depends upon the previous byte of checksum, so the checksum computation cannot be parallelized.

SWATT is considered to be weak in certain aspects which include: It is not hardware specific i.e the attestation of code does not include information about the hardware, the protocol has to be initiated by the Verifier on suspicion, it does not provide implicit protection for every response sent by the Sensor Node, TOCTTOU (Time of Check to Time of Use) is different so the attacker can change the memory contents of the sensor node after the verification and before the time of use, and lastly due to transmission of challenge and then the response on the network, the verification procedure is subject to Rainbow and Interference Attack.

In a Rainbow Attack an attacker intercepts the challenge and response messages transmitted between the verifier and a sensor node[49]. The interception of these messages can infer the attestation information like length of the challenge to the attacker. Hence the attacker then sends fake challenges to the sensor nodes and can build up a lookup table consisting of challenges and their respective responses, which can be later used for false attestation.

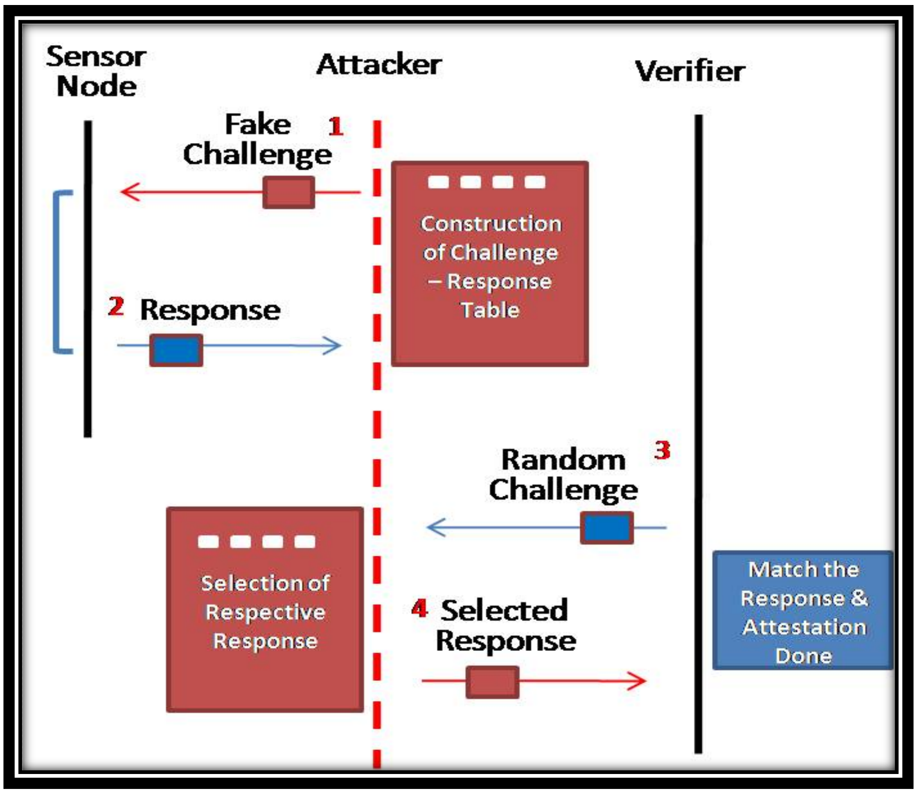

Sensor Node
Attacker
Verifier
Fake Challenge
1
Construction of Challenge – Response Table
2
Response
Random Challenge
3
Selection of Respective Response
4
Selected Response
Match the Response & Attestation Done

Figure-3.1 Rainbow Attack

Whereas in an Interference Attack, the attacker's aim is to induce the sensor node to send mis-calculated responses [16].

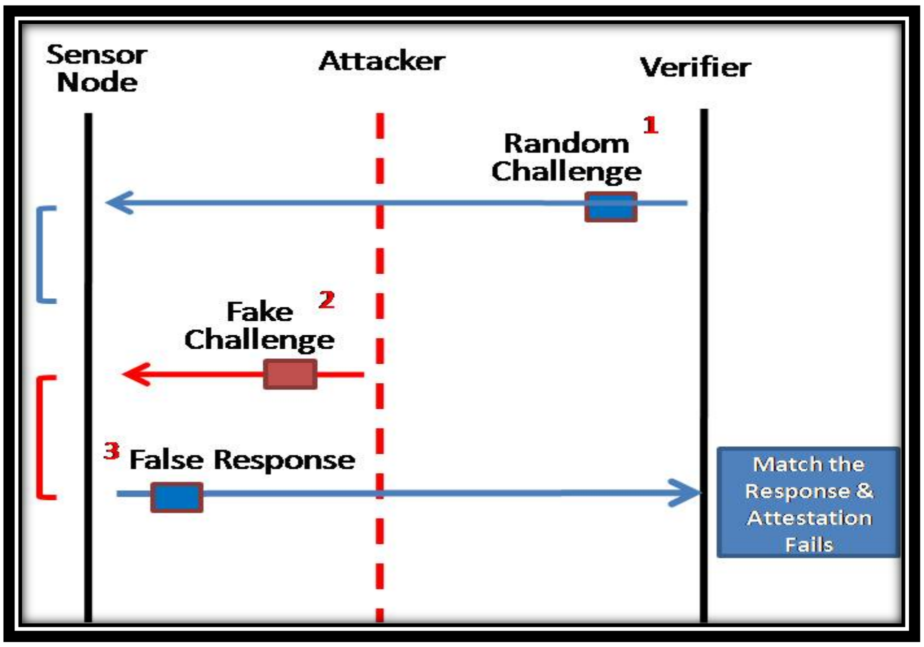


Figure-3.2 Interference Attack

3.2.2 Distributed Software Based Attestation

Distributed Software Based Attestation for Node Compromise. It is also a challenge response protocol, which proposes two different schemes [23]. Scheme-1: A Basic Threshold Secret Sharing Scheme (BTSS), and Scheme-2: Majority Voting Based Attestation Scheme (MVBS).

3.2.2.1Basic Threshold Secret Sharing Scheme-1

This scheme consists of three steps. Step-1 is noise generation in which before deployment, empty memory space of each sensor node say 's', is filled with random noise using a PRG. Step-2 is about secret share distribution. In this step, after deployment sensor node discovers its neighbors by sending "Hello" message and starts a timer which expires after "t_{min}". It establishespair wise key with each neighbor and then splits noise generation seed "Su" into multiple shares and send a separate share to each of its neighbors.When the timer" t_{min}" expires it removes "Su" from its memory. In Step-3attestation is triggered if more than "half" of neighbors detect the abnormal behavior of say node”u”. Then all the neighbors select a cluster head V_h.An authentication challenge **‘**R_n**’** is sent by V_h to node’u’.While node u computes challenge “C” over its memory using ‘R_n’, the cluster head V_h collects ‘K’ secret shares from neighbors of ‘u’ and recovers the noise generation secret seed S_u.The secret share S_u recovered, is verified by taking hash and comparing with $H(S_u)$ received earlier from node u.If the recovered S_u is not verified then the cluster head collect another k shares which does impose a computation overhead.After the matched $S_{u,}$ the cluster head V_h computes the expected response C_{exp} from R_n and receives the response from the node u and compare the both.

In this scheme, cluster head needs to be a trusted party, which performs the code attestation of end nodes. A compromised neighbor may contribute a false share so cluster head will need to select another k shares from the sensor nodes in order to recover the correct seed. In this case additional energy will be consumed. To counter this each node should store a copy of hash of each share also. But it will consume extra memory for each sensor node.Attestation is based on random challenge so another node cannot pre-compute the response. Attacker has to obtain $>$= k shares to recover the secret seed. The time at which end node is tested is different than the time when it is used.

3.2.2.2 Majority Voting Based Attestation Scheme -2

It is also a challenge response based scheme that consists of 3 steps. Step-1 is information distribution. In this step, before deployment each node is pre-loaded with Pseudo random noise, and also with n tuples of (C_i,R_i) (where n is number of expected neighbors and (C_i,R_i) is the challenge and corresponding response pairs).Each tuple is generated by an offline server and after deploymentevery node say u discovers its neighbors. Securely delivers each neighbor a randomly picked tuple of (C_i,R_i) within time t_{min} and erases all the tuples from its memory after $t_{min.}$

Step-2 is about attestation. If half of the neighbors agree to attest node u, they do it in a sequence.Finally if the number of neighbors with negative opinions exceed (n+1)/2 i.e more than half, Node'u' will be identified as a compromised node.In step-3the attested node computes the response to the challenge by making $\mathbf{i_t}$ (Number of iterations) = **(m ln m) / bn** accesses to the memory. The point in question is why we have bn in the denominator ? it is because we have C_i distinct challenges so all the neighbors may corporately traverse each memory cell of node'**u**' at least once.

Due to majority voting this scheme is vulnerable to good mouth and bad mouth attacks [26], in case we already have some compromised nodes among the immediate neighbors of an attested node. The time at which end node is tested is different than the time when it is used. Code attestation is initiated when more than half of the neighbors detect a suspicious behavior of any node, which is totally subtle and will have an adverse effect on energy consumption as well as the bandwidth utilization, whenever a code test procedure is initiated.If in scheme 2, i_t = (m ln m)/b, then the computation cost in node'u' will be n times the cost of scheme 1, as in scheme 2, node u has to compute R_n responses to $C_n$challenges.

3.2.3OMAP-One Way Memory Attestation Protocol

This software based one way attestation protocol was implemented on a 8 bit MPU based smart meter for remote code attestation [24]. The smart meter had 64 KB flash memory and 5 KB RAM.The protocol is based upon certain assumptions about the verifier, smart meter and the attacker. The assumptions about the verifier include: verifier knows, the exact specifications of end device's hardware, exact configuration of remote device memory, verifier maintains an exact copy of the memory of remote device and lastly verifier cannot be compromised by the attacker.Similarly the assumptions about the smart meter include: Smart Meter utilizes the Serial Number (SN) for attestation, attacker cannot modify, or forge the SN and every smart meter has a unique SN. The assumptions for the attacker are that, he can `modify the memory contents,` can eavesdrop all data transmitted over the Advance Metering Network (AMI), cannot replace smart meter hardware i.ecannot change BIOS of smart meter,cannot add a memory,cannot change memory access timing andcannot increase clock speed of the processor.

3.2.3.1 OMAP Algorithm

Seed Generation ($W_{k)}$

It is achieved by taking hash of time $\mathbf{t_k}$and secure serial number (**SN**) of the smart meter. The time parameter is used to avoid replay and checksum pre-computation attacks. Whereas **k** is the number of accesses to be made to the program memory. For the analysis purpose the author has implemented **k** for 20 iterations.

$$\mathbf{W_k = H(t_k , SN)}$$

Address Generation

In order to ensure that the contents are extracted from a random portion of the memory, a Pseudo Random Generator is used in the form of RC-4 Cipher. It takes 32 bits as an input and also gives out 32 bits output. $\mathbf{A_k}$ is the 16 bit memory address.

$$\mathbf{A_k = (RC4\ (W_{k)})}$$

Content Extraction

The function **Fn** extracts contents from the 16 bit memory address. In 20 iterations 40 bytes of contents are extracted i.e 80 Hexa Decimal Numbers.

$\mathbf{Q_k = Fn(A_k)}$

Message Construction

Message m is constructed by concatenating contents extracted in each iteration.

$$\mathbf{m = Q_1 // Q_2 // Q_3 // Q_4 //\ldots\ldots\ldots// Q_N}$$

Checksum Transmission

The message that is transmitted by the smart meter to the base station / controller consists of Hash of message **H(m)** and the time $\mathbf{t_k}$.

Send {**H(m),**$\mathbf{t_k}$)}

Checksum Verification

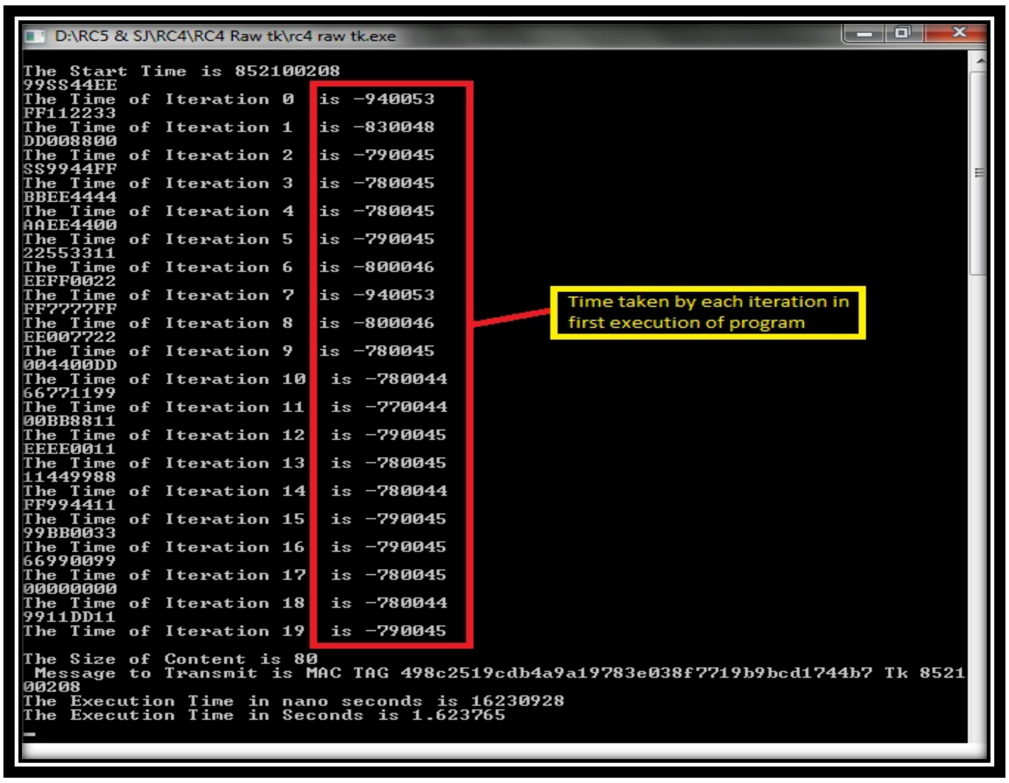
The base station / controller is the verifier for the code attestation of smart meter. The verifier takes the time $\mathbf{t_k}$ from the received message and using the image of the memory of the smart meter, computes the checksum **H'(m)** and compares it with the received **H(m)**.

3.2.3.2 Observations on OMAP

The positive aspects of OMAP include: attacker cannot forge a message, because of use of secret SN and time tk.Attacker cannot compute all the iterations i.e from 1 to N, in parallel because of time t_k.OMAP is not a challenge – response protocol so it avoids network attacks like, Rainbow Attack and Interference Attack.In order to check the efficacy of OMAP, the algorithm with two components, a verifier and a smart meter were implemented on a single computer; Intel Core 2 Duo T5550 1.8 GHz, with the simulation of memory size of 128KB.

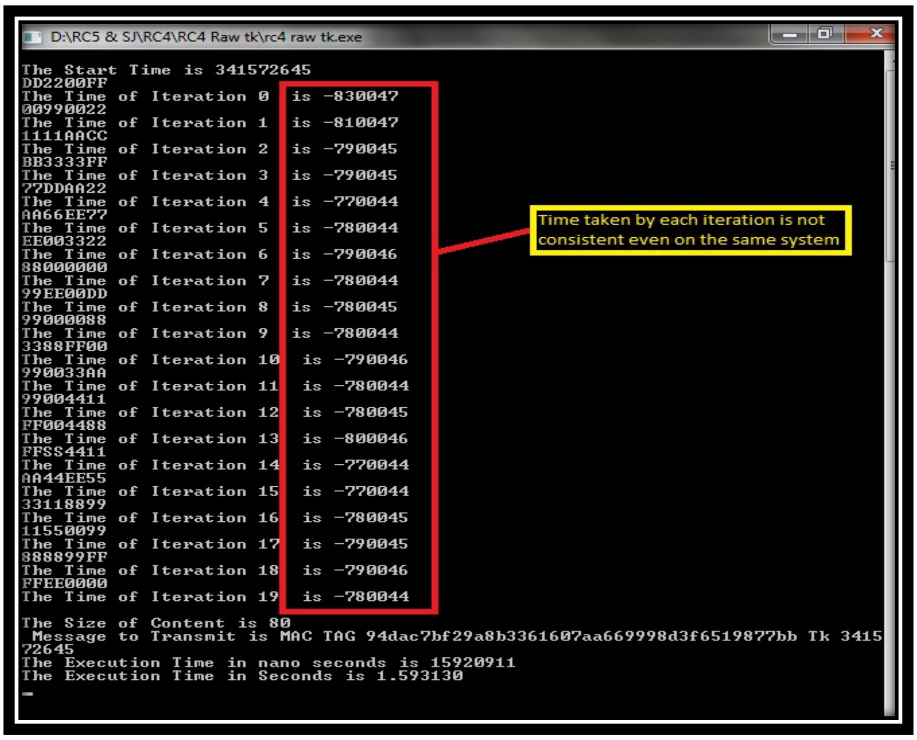
Although OMAP is claimed it to be a novel, remote, and one way code attestation procedure that avoids network attacks, yet it fails to present a distinct initiation of code attestation procedure. Such a severe weakness can put a tag of inefficiency and vulnerability to any algorithm irrespective of its computational efficacy and security.

The major observation on OMAP Algorithm is the use of time $\mathbf{t_k}$ for seed generation in each iteration. The verifier also uses the same time for seed generation while computing the checksum for verification. Such a dependence on time requires very efficient time synchronization, which is very difficult to achieve in practice. The OMAP Algorithm was implemented in Dev C++ and it was observed that even on a single system the execution time for each iteration of algorithm was not consistent. The screen shots of time measurements shown in Figure-3.3 and Figure-3.4 clearly indicate that there is a sound difference and inconsistency in time taken by each iteration of the algorithm which ultimately leads to a checksum mismatch at the verifier.


D:\RC5 & SJ\RC4\RC4 Raw tk\rc4 raw tk.exe
Time taken by each iteration in first execution of program

```
The Start Time is 852100208
99SS44EE
The Time of Iteration 0  is -940053
FF112233
The Time of Iteration 1  is -830048
DD008800
The Time of Iteration 2  is -790045
SS9944FF
The Time of Iteration 3  is -780045
BBEE4444
The Time of Iteration 4  is -780045
AAEE4400
The Time of Iteration 5  is -790045
22553311
The Time of Iteration 6  is -800046
EEFF0022
The Time of Iteration 7  is -940053
FF7777FF
The Time of Iteration 8  is -800046
EE007722
The Time of Iteration 9  is -780045
004400DD
The Time of Iteration 10  is -780044
66771199
The Time of Iteration 11  is -770044
00BB8811
The Time of Iteration 12  is -790045
EEEE0011
The Time of Iteration 13  is -780045
11449988
The Time of Iteration 14  is -780044
FF994411
The Time of Iteration 15  is -790045
99BB0033
The Time of Iteration 16  is -790045
66990099
The Time of Iteration 17  is -780045
00000000
The Time of Iteration 18  is -780044
9911DD11
The Time of Iteration 19  is -790045

The Size of Content is 80
 Message to Transmit is MAC TAG 498c2519cdb4a9a19783e038f7719b9bcd1744b7 Tk 8521
00208
The Execution Time in nano seconds is 16230928
The Execution Time in Seconds is 1.623765
_
```

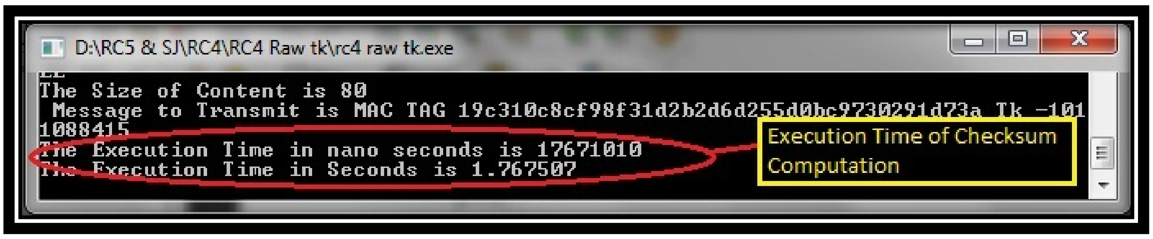
Figure-3.3OMAP-Time measurement of first execution


D:\RC5 & SJ\RC4\RC4 Raw tk\rc4 raw tk.exe
Time taken by each iteration is not consistent even on the same system

```
The Start Time is 341572645
DD2200FF
The Time of Iteration 0  is -830047
00990022
The Time of Iteration 1  is -810047
1111AACC
The Time of Iteration 2  is -790045
BB3333FF
The Time of Iteration 3  is -790045
77DDAA22
The Time of Iteration 4  is -770044
AA66EE77
The Time of Iteration 5  is -780044
EE003322
The Time of Iteration 6  is -790046
88000000
The Time of Iteration 7  is -780044
99EE00DD
The Time of Iteration 8  is -780045
99000088
The Time of Iteration 9  is -780044
3388FF00
The Time of Iteration 10  is -790046
990033AA
The Time of Iteration 11  is -780044
99004411
The Time of Iteration 12  is -780045
FF004488
The Time of Iteration 13  is -800046
FFSS4411
The Time of Iteration 14  is -770044
AA44EE55
The Time of Iteration 15  is -770044
33118899
The Time of Iteration 16  is -780045
11550099
The Time of Iteration 17  is -790045
888899FF
The Time of Iteration 18  is -790046
FFEE0000
The Time of Iteration 19  is -780044

The Size of Content is 80
 Message to Transmit is MAC TAG 94dac7bf29a8b3361607aa669998d3f6519877bb Tk 3415
72645
The Execution Time in nano seconds is 15920911
The Execution Time in Seconds is 1.593130
_
```

Figure-3.4OMAP-Time measurement of second execution

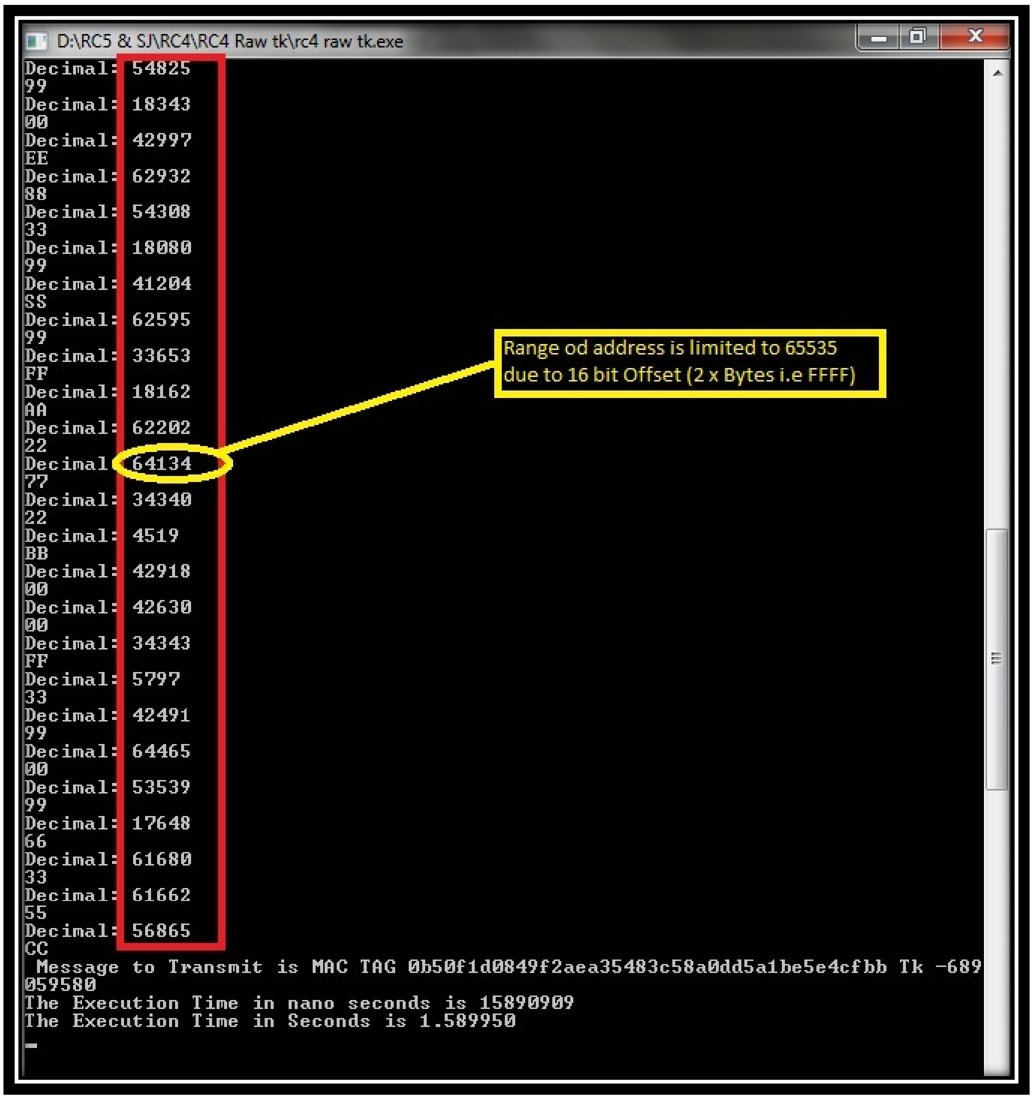
The OMAP Algorithm performs code attestation independent of routine smart meter application therefore in order to ensure integrity of program memory it extracts contents from the program memory by accessing the memory 20 times.A screen shot of execution time for checksum computation is shown in figure-3.5. The long execution time is due to the fact that attestation protocol is independent to routine sensor update and is not performed at run time, therefore the physical memory of the sensor mote is being accessed for 20 times. Resultantly the computation time of the protocol algorithm is around 1.7 seconds.


D:\RC5 & SJ\RC4\RC4 Raw tk\rc4 raw tk.exe
Execution Time of Checksum Computation

```
The Size of Content is 80
 Message to Transmit is MAC TAG 19c310c8cf98f31d2b2d6d255d0bc9730291d73a Tk -101
1088415
The Execution Time in nano seconds is 17671010
The Execution Time in Seconds is 1.767507
```

Figure-3.5OMAP-Execution time

The 32 bit output of RC-4 Cipher (PRG) is used to construct two 16 bit memory addresses for content extraction. The use of 16 bit address limits the memory address range to ffff thus restricting the random access to memory of size not exceeding 65535 Bytes (Approximately 64KB). Figure 3.6 shows the screen shot of range of 16 bit memory addresses.

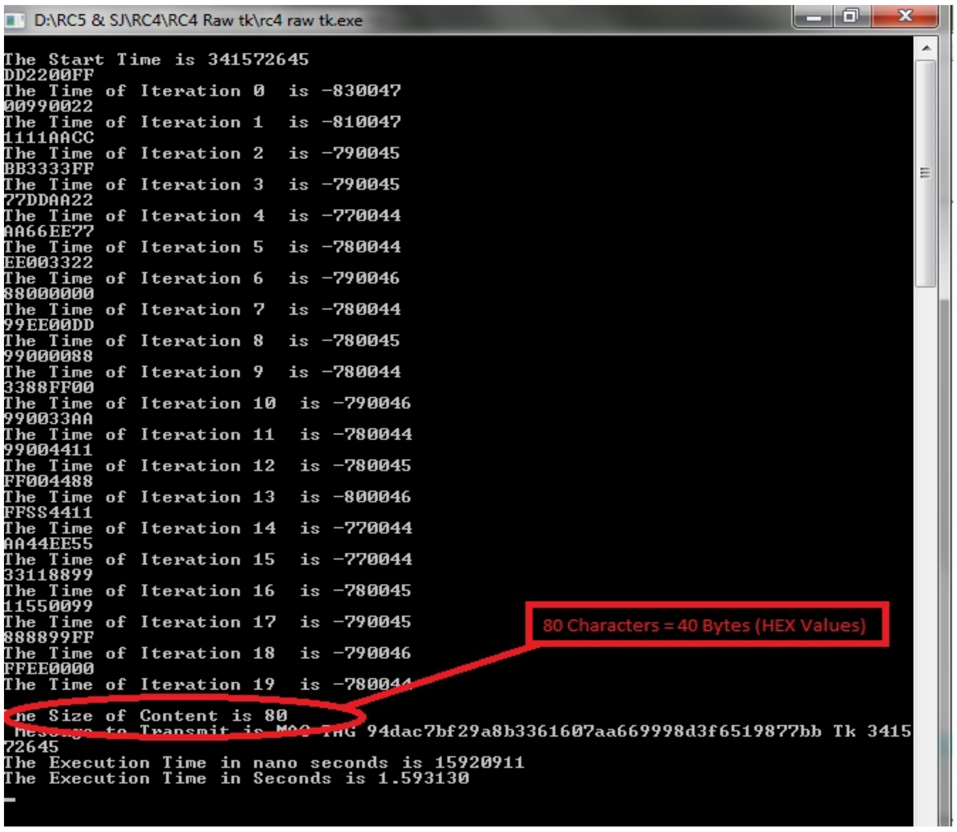


D:\RC5 & SJ\RC4\RC4 Raw tk\rc4 raw tk.exe
Range od address is limited to 65535 due to 16 bit Offset (2 x Bytes i.e FFFF)

```
Decimal: 54825
99
Decimal: 18343
00
Decimal: 42997
EE
Decimal: 62932
88
Decimal: 54308
33
Decimal: 18080
99
Decimal: 41204
SS
Decimal: 62595
99
Decimal: 33653
FF
Decimal: 18162
AA
Decimal: 62202
22
Decimal: 64134
77
Decimal: 34340
22
Decimal: 4519
BB
Decimal: 42918
00
Decimal: 42630
00
Decimal: 34343
FF
Decimal: 5797
33
Decimal: 42491
99
Decimal: 64465
00
Decimal: 53539
99
Decimal: 17648
66
Decimal: 61680
33
Decimal: 61662
55
Decimal: 56865
CC
 Message to Transmit is MAC TAG 0b50f1d0849f2aea35483c58a0dd5a1be5e4cfbb Tk -689
059580
The Execution Time in nano seconds is 15890909
The Execution Time in Seconds is 1.589950
```

Figure-3.6 Range of 16 Bit Memory Addresses

The generation of two addresses in each iteration limits the number of extracted contents to 2 bytes and the total number of extracted memory contents is 40 Bytes (Figure-3.7). Which if compare to the execution time of checksum computation is very less. It will be shown in later analysis that detection rate of modified code is directly proportional to the number of extracted contents. The more are the extracted memory contents the more is the probability of detection of modified memory.


D:\RC5 & SJ\RC4\RC4 Raw tk\rc4 raw tk.exe
80 Characters = 40 Bytes (HEX Values)

```
The Start Time is 341572645
DD2200FF
The Time of Iteration 0  is -830047
00990022
The Time of Iteration 1  is -810047
1111AACC
The Time of Iteration 2  is -790045
BB3333FF
The Time of Iteration 3  is -790045
77DDAA22
The Time of Iteration 4  is -770044
AA66EE77
The Time of Iteration 5  is -780044
EE003322
The Time of Iteration 6  is -790046
88000000
The Time of Iteration 7  is -780044
99EE00DD
The Time of Iteration 8  is -780045
99000088
The Time of Iteration 9  is -780044
3388FF00
The Time of Iteration 10  is -790046
990033AA
The Time of Iteration 11  is -780044
99004411
The Time of Iteration 12  is -780045
FF004488
The Time of Iteration 13  is -800046
FFSS4411
The Time of Iteration 14  is -770044
AA44EE55
The Time of Iteration 15  is -770044
33118899
The Time of Iteration 16  is -780045
11550099
The Time of Iteration 17  is -790045
888899FF
The Time of Iteration 18  is -790046
FFEE0000
The Time of Iteration 19  is -780044

ne Size of Content is 80
Message to Transmit is MAC TAG 94dac7bf29a8b3361607aa669998d3f6519877bb Tk 3415
72645
The Execution Time in nano seconds is 15920911
The Execution Time in Seconds is 1.593130
_
```

Figure-3.7 Size of Extracted Memory Contents

Dongwon Seo claims in [24] that OMAP detects a 20% modified memory with a 95% probability. Considering the optimized size of malware now a days and a memory size of 128 K Bytes, even 15% modified memory is enough to keep a malware. Therefore detection rate of 95% for a 20% modified memory is quite low. Moreover, if end device's Serial Number is exposed to an attacker, he can compute acorrect checksum and can impersonate a device and send false readings.Another major weakness is that the time at which end node is tested is different than the time when it is used.

3.3 Summary

Code Attestation is considered to be the most potent defense against insider attacks including all dimensions of Sybil Attack. Code Attestation can be based upon a challenge response or a one way attestation protocol. However the challenge – response protocol is susceptible to network attacks. The other weaknesses found in existing code attestation schemes are difference in TOCTTOU, elusive start of code test procedure, and independence of code test procedure to routine application running on a wireless sensor node. The difference in TOCTTOU can render protection provided by any code attestation scheme ineffective, in case attacker exploits this time gap and make any modification in the code just before the wireless sensor is suppose to send its scheduled updates to base station or sink node. The independent code attestation procedures result in excessive bit transmission over and above bits transmitted for sensor reading updates. This excessive bit transmission exhausts the available energy on a sensor node thus reduces its lifetime. These flaws make it necessary to develop an efficient and a secure code attestation scheme.

Chapter 4

One Way Code Attestation Protocol (OWCAP)

4.1 Introduction

OWCAP an extension of OMAP (explained in previous chapter) is the product of this research. It has been developed to improve upon the weaknesses of existing noteworthy code attestation protocols. It is not only an efficient code attestation protocol but also provide requisite security against insider / integrity attacks. Although in this research the main focus is on code attestation in WSN, but the same technique can be exported to any ICS that relies on information sharing between sensors, actuators and the controllers. In this chapter a comprehensive explanation of OWCAP algorithm is given, followed by the progress it has made to improve upon the observations of OMAP and in the end a detailed analysis of OWCAP is carried out covering both, the performance and the security aspects.

4.2 Objectives of OWCAP

The weaknesses in OMAP and previously discussed challenge response protocols led to achieve certain objectives for the development of OWCAP.

1. Tangible start of code attestation procedure.
2. Efficient and reliable seed generation.
3. Minimal execution time of checksum computation.
4. Increased detection rate of modified memory.
5. Prevent pre-computation and parallel computation of memory checksum.
6. Message Integrity.
7. Implicit code attestation.
8. Minimal transmission overhead.

9. Diminish TOCTTOU gap.
10. Avoid network attacks like rainbow, interference, message modification and impersonation attacks.
11. Minimal energy consumption.

4.3 Assumptions

OWCAP Protocol is based on certain assumptions. For an attested sensor node the assumptions include: The serial number of a sensor node is secret, which cannot be disclosed or forged even upon physical compromise of the sensor node[24]. OWCAP utilizes the serial number in seed generation for pseudo random memory traversal. The assumptions for the verifier are that; Base station or the sink node is in clear picture of the attested node's memory size and processing power. The verifier also keeps the memory images of its respected end nodes, in order to utilize these during code attestation process. In the end the assumptions for the attacker are that; Attacker can read and write into the memory (containing code) of a sensor node. Thus an attacker is capable of injecting a malicious code into empty area of sensor node's memory. The attacker can also eavesdrop and modify the messages transmitted over the wireless sensors networks.

4.4 OWCAP Algorithm

OWCAP is developed to improve upon the weaknesses of OMAP and further increase the reliability, effectiveness and efficiency of the Code Attestation Protocol. In order to ensure seamless code attestation and to avoid TOCTTOU Attack, OWCAP provides implicit code attestation along with routine transmission of sensor reading.

4.4.1 Seed Generation

The attested sensor node computes the seed for the PRG (Pseudo Random number Generator) by taking hash of current time and the secret serial number of the node. The use of current time, act as a nonce and avoids replay attacks. Whereas the use of serial number avoids impersonation attacks. The serial number can be kept secure using white box cryptography already being researched upon extensively for digital rights management (DRM). The algorithm of seed generation is as under:-

```
Initialize integer j;
$T_s$;          //Time stampindicating starting time of the algorithm used as a Nonce
                    to avoid replay and pre-computation attacks.
Fp;             //Image of the present program memory.
Counter;        //Its value is same as k below, to provide randomness of
                    the seed.
k=1 to 20;      // The memory is accessed 20 times same as OMAP
W_k = H($T_s$, S_N, Counter)  // S_N is the secure hardware serial number of
                                   the sensor node.
                // W_k is the seed for PRG.
```

4.4.2 Pseudo Random Memory Traversal

The seed generated in first step is the input of PRG, which can be an RC4, RC5 or Skipjack Cipher. These ciphers are selected based on the performance analysis of various block and stream ciphers in [29] and [30], but the use of RC-5 and Skipjack cipher is recommended since they have already been implemented in Tinysec [39]for message encryption and message authentication code. Therefore using the same cipher as a PRG saves upon precious code size. The pseudo random memory traverse prevents pre-computation and parallel computation of checksum by XORing last round's value of memory contents with the current round's value.

```
[A_0,A_1,……A_31] = RC4(W_k);
        // [A_0-A_31] are the memory addressesgenerated as the output of
          PRG which can be RC4, RC5 and Skipjack Cipher or simple SHA2
Q_k = Mem(A_0), Mem(A_1), Mem(A_2)……Mem(A_31)];
Q   =  $Q_0$ // $Q_1$ // $Q_2$ //……………………// $Q_{19}$
 Q carries memory contents.
```

4.4.3 Message Construction

This is the most important part of OWCAP. The initial message P is constructed by appending sensor update 16 bit value of sensor reading S_R with the extracted memory contents (for memory size of 128 KB). An HMAC is then computed over P. The message to be sent to the verifier is [HMAC(P), S_R, Ts]. The HMAC computed over message P, not only provides integrity of sensor update S_R but also provides implicit code attestation. In normal condition the message sent to the verifier is MAC(S_R) and S_R, so the only increase in transmission overhead in case of OWCAP (in plain mode) is time Ts which is required by the verifier for seed generation.

```
P = [Q // S_R // Ts];
Checksum = MAC (P);
Message Send =  [MAC(P), S_R, t_k];
//MAC is Message Authentication Code which takes an arbitrary number of input
```

4.4.4Verification

After receiving the message [MAC(P), S_R, Ts] from attested node, the verifier, using the time Ts, S_R and the already stored serial number S_N of the respected end node, computes the expected message authentication code. The verifier then compares both the message authentication codes for verification. In this step, OWCAP provides implicit code attestation along with the integrity check of sensor update S_R thus eliminating TOCTTOU gap.

$$MAC'(P) = MAC(P)$$

4.4.5 Recovery and Revocation

In case the expected MAC'(P) does not match the received MAC(P), the verifier increments a counter value for respective node until it reaches its maximum threshold of 3. Threshold is set to 3 to avoid false rejection rate at the very first instance. As soon as the counter value for an end node reaches 4 the node is immediately black listed and no further messages are accepted from or sent to this node. The node revocation is immediately followed by an end node code update procedure. This can be done remotely; if possible, otherwise a physical end node code update procedure is performed.

4.5 Improving Upon the Observations of OMAP

4.5.1 Implicit Initiation of Attestation Procedure

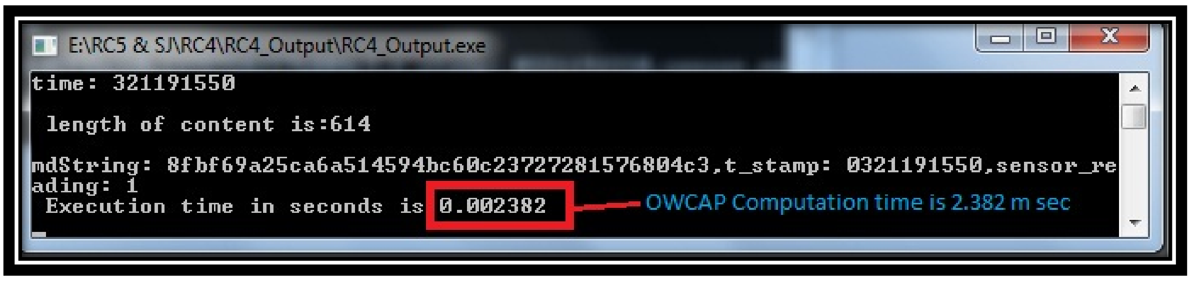
In order to circumvent the threats and vulnerabilities like TOCTTOU attack, ROP and vague initiation of code attestation, OWCAP provides implicit code attestation with each sensor update. Such a protocol completely evades the question of when to initiate the code attestation procedure. A detailed comparative analysis of all the code attestation schemes will be presented in the next chapter.

4.5.2 Seed Generation and Time Synchronization

In order to avoid problems related to time synchronization during seed generation at the sensor node as well as at the verifier end, the algorithm is improved to achieve seamless seed generation. Only the time of start of code attestation, procedure Ts is used in the seed generation to prevent replay and checksum pre computation attacks, and to achieve randomness of seed, a counter value is used and incremented for all the iterations of seed generation.

4.5.3 Execution Time of Checksum Computation

In difference to OMAP which took a considerable time to access the memory for just 20 iterations, OWCAP is developed to save upon the precious time by a very large factor. A screen shot of execution time for checksum computation using RC4 Cipher is shown in figure-4.1. It is achieved by altering the code attestation protocol from explicit to implicit attestation during a routine sensor application. Each time at the start of code attestation procedure the image of program memory of the sensor node is taken and stored in a buffer at runtime then in each iteration the contents of the memory are acquired randomly from the buffer rather than physically reading the memory every time. Whereas same technique is not possible to implement in OMAP or previous code attestation schemes due to TOCTTOU gap.


E:\RC5 & SJ\RC4\RC4_Output\RC4_Output.exe

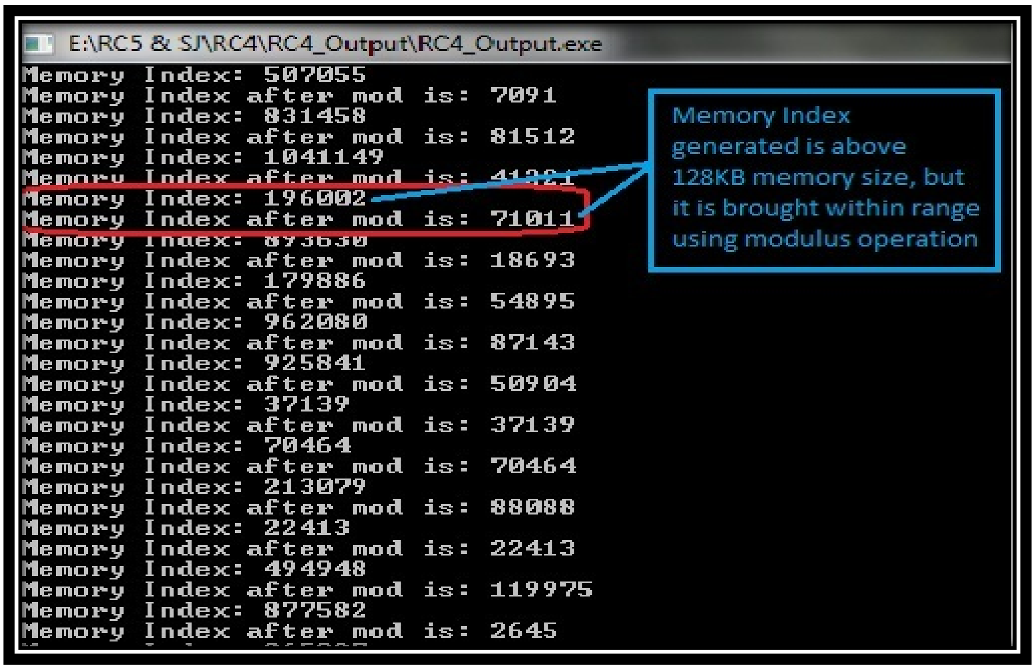
```
time: 321191550

 length of content is:614

mdString: 8fbf69a25ca6a514594bc60c23727281576804c3,t_stamp: 0321191550,sensor_re
ading: 1
 Execution time in seconds is 0.002382
```

OWCAP Computation time is 2.382 m sec

Figure-4.1 Reduced Execution Time of OWCAP

4.5.4 Range of Addresses

OWCAP is developed to overcome the problem of small range of memory addresses and it can now access a memory size of up to 16.77 MB. In order to limit the generated memory index (address) within the maximum size of the file, a prime modulus operation is introduced. For example, for a memory size of 128 KB the mod of a prime number (127997) closest to 128 K is taken. A glimpse of range of addresses generated before and after the modulus operation is shown in figure -4.2.

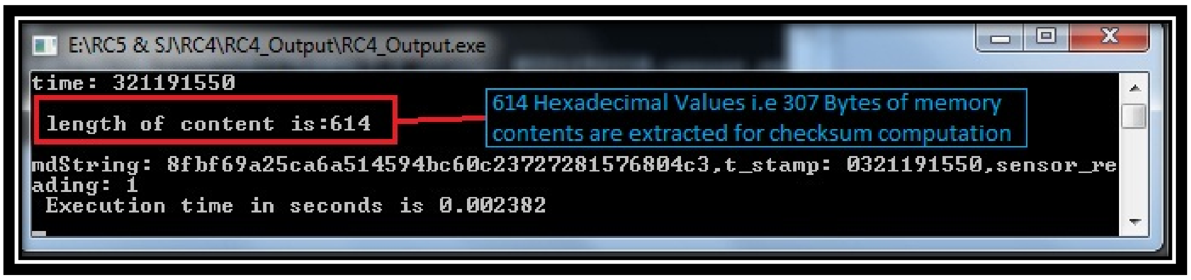

E:\RC5 & SJ\RC4\RC4_Output\RC4_Output.exe
Memory Index generated is above 128KB memory size, but it is brought within range using modulus operation

```
Memory Index: 507055
Memory Index after mod is: 7091
Memory Index: 831458
Memory Index after mod is: 81512
Memory Index: 1041149
Memory Index after mod is: 41321
Memory Index: 196002
Memory Index after mod is: 71011
Memory Index: 893630
Memory Index after mod is: 18693
Memory Index: 179886
Memory Index after mod is: 54895
Memory Index: 962080
Memory Index after mod is: 87143
Memory Index: 925841
Memory Index after mod is: 50904
Memory Index: 37139
Memory Index after mod is: 37139
Memory Index: 70464
Memory Index after mod is: 70464
Memory Index: 213079
Memory Index after mod is: 88088
Memory Index: 22413
Memory Index after mod is: 22413
Memory Index: 494948
Memory Index after mod is: 119975
Memory Index: 877582
Memory Index after mod is: 2645
```

Figure-4.2 OWCAP Range of addresses

4.5.5 Number of Extracted Contents

OWCAP not only provides computational efficiency but has also increased the detection rate by taking checksum of 5 times more memory contents than in OMAP. In OMAP the detection of memory modification is based on a checksum of 80 Bytes of contents whereas in a considerably less time OWCAP provides better detection rate by computing checksum over 307 Bytes (Figure -4.3) of memory contents. There can be max 320 Bytes of extracted contents.



```
E:\RC5 & SJ\RC4\RC4_Output\RC4_Output.exe
time: 321191550
 length of content is:614
mdString: 8fbf69a25ca6a514594bc60c23727281576804c3,t_stamp: 0321191550,sensor_re
ading: 1
 Execution time in seconds is 0.002382
```

614 Hexadecimal Values i.e 307 Bytes of memory contents are extracted for checksum computation

Figure-4.3 Size of memory contents extracted by OWCAP

4.5.6OWCAP Message Transmission

The sequence and number of messages to be sent in OWCAP are shown in figure-4.4

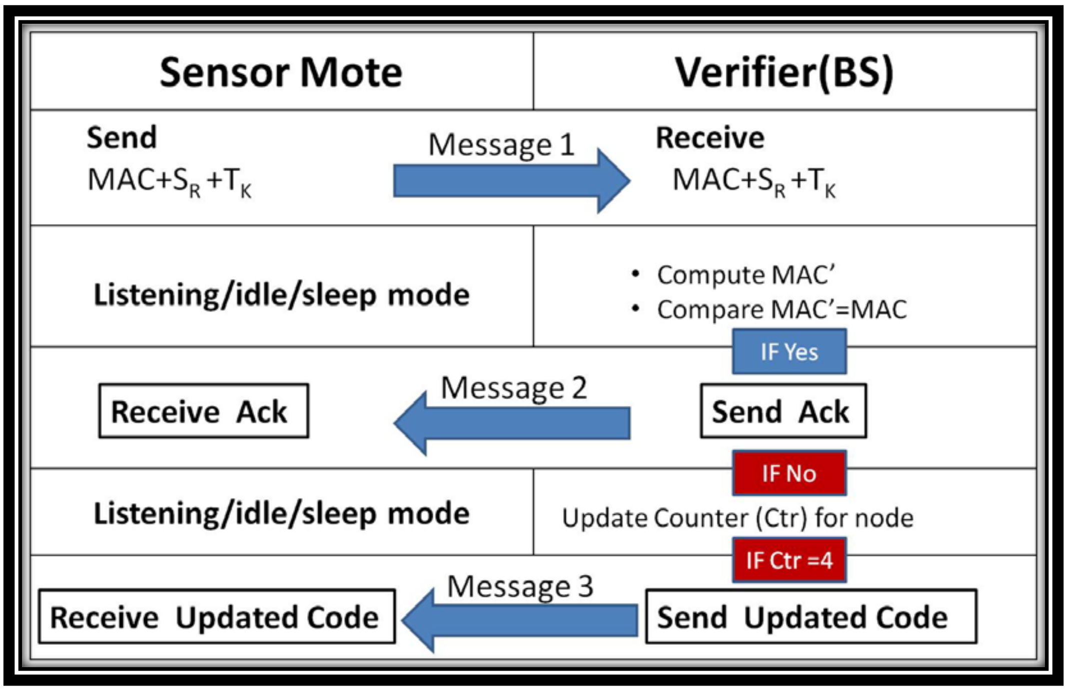


Figure-4.4 OWCAP Message Transmission

4.6 Performance Analysis of OWCAP

OWCAP is implemented in Dev C++ with few variants in order to gauge its efficiency and effectiveness by measuring execution time and code modification detection rate. Different options of using various stream and block ciphers including RC-4, Skipjack and RC-5 as PRG like were exploited. The exec time is measured for different memory sizes including 256 KB, 128 KB, 64 KB and 32 KB. In order to simulate a memory size, files of different sizes as mentioned above were created in Visual Studio 2008. The code for each variant of OWCAP and file creation is mentioned at Appendix A, B, C, D, E, F, G and H respectively. The execution time and the detection rate is inversely and directly proportional respectively to the number of accesses to the program memory. As per Coupon Collector’s Problem the code attestation procedure does m * ln (m) accesses to the memory in order to access each memory location at least once, where m is size of the memory. The actual number of the memory accesses required vis a vis the memory accesses implemented in the simulation are as shown in Table 4.1.

Table 4.1 Number of Memory Accesses

Ser	Memory Size	Actual Number of Memory Accesses Required as per m*ln(m)	Number of Memory Accesses Simulated
1.	256 KB	31,87,951	40
2.	128 KB	15,05,253	20
3.	64 KB	7,08,265	10
4.	32 KB	3,31,952	5

4.6.1 OWCAP Execution Time Measurements

The time measurements of all the variants of OWCAP are taken as per simulated number of memory accesses mentioned in Table 4.1. The time measurements clearly show that OWCAP variant using SHA-2 hash function as a PRG has the least time for all memory sizes. As per [34] we can use a Hash function as a DRNG (Deterministic Random Number Generator), and also as per [35] different gambling services also use Hash functions for selecting a random winner. The graphical representation of average execution time comparison IRO all the variants of OWCAP is shown in figure-4.5 and detailed results are shown below:-

Table 4.2 Execution Time

Ser	Type of PRG	Memory Size	Execution Time (msec / Iteration for 20 Memory Accesses)										Avg Time (msec)
			1	2	3	4	5	6	7	8	9	10	
1	**SHA-256**	256 KB	2.704	2.550	2.817	2.570	2.576	2.503	2.573	2.571	2.636	2.513	**2.601**
		128 KB	2.106	2.106	2.056	2.254	1.927	1.917	2.243	2.754	2.135	2.057	**2.155**
		64 KB	1.387	1.355	1.366	1.346	1.345	1.364	1.387	1.333	1.395	1.376	**1.365**
		32 KB	1.128	1.131	1.152	1.134	1.157	1.158	1.163	1.138	1.134	1.255	**1.155**
2	**RC-4**	256 KB	4.847	4.634	3.870	5.589	3.480	2.959	4.672	2.969	3.210	3.031	**3.926**
		128 KB	2.206	2.165	2.134	2.113	2.170	4.925	2.132	2.290	2.167	7.045	**2.934**
		64 KB	1.835	1.539	1.512	1.497	1.579	1.735	1.724	1.679	1.732	1.528	**1.640**
		32 KB	1.298	1.248	1.252	1.264	1.334	1.332	1.341	1.296	1.287	1.577	**1.322**
3	**RC-5**	256 KB	2.931	4.921	3.012	3.027	2.978	2.919	3.117	3.046	3.297	3.213	**3.246**
		128 KB	2.944	2.204	2.267	2.341	2.247	2.518	2.872	2.200	2.213	2.493	**2.429**
		64 KB	1.616	1.703	1.677	1.696	1.605	1.680	1.630	1.920	1.691	1.840	**1.706**
		32 KB	1.391	1.388	1.417	1.406	1.587	1.346	1.526	1.446	1.565	1.426	**1.450**
4	**Skipjack**	256 KB	2.895	2.807	2.814	2.816	3.047	3.052	2.966	2.814	2.878	2.925	**2.901**
		128 KB	2.486	2.376	2.435	2.522	2.636	2.461	2.494	2.505	2.429	2.342	**2.468**
		64 KB	1.563	1.472	1.470	1.460	1.521	1.530	1.551	1.691	1.542	1.479	**1.527**
		32 KB	1.436	1.138	1.207	1.198	1.243	1.160	1.160	1.238	1.182	1.229	**1.219**

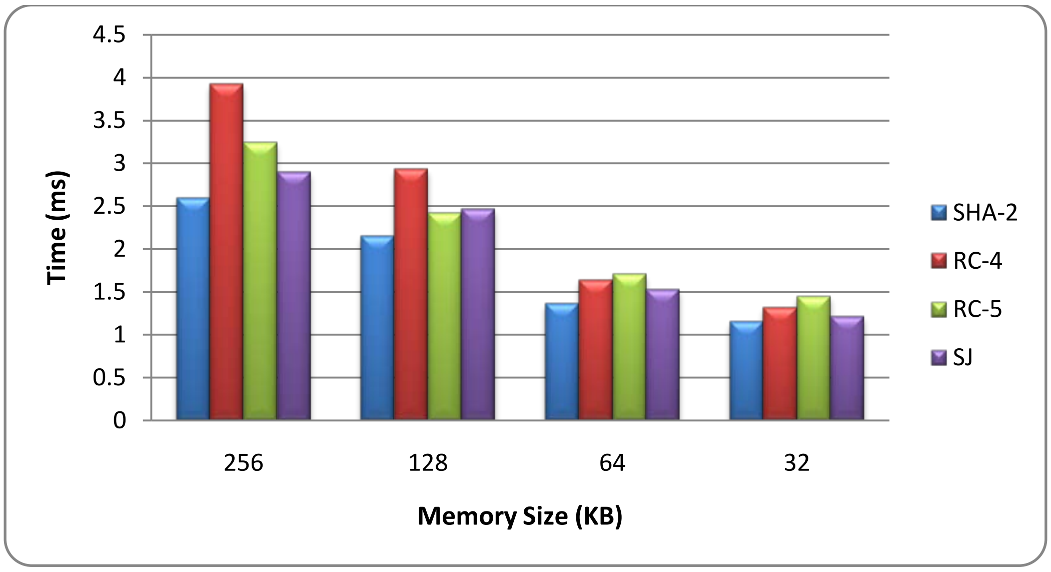


Figure-4.5 Comparison of average execution time

4.6.2 Memory Detection Rate

Detection rate of modified memory was measured at various sizes of malicious memory for all the variants of OWCAP. The memory size was fixed to 128 KB and the size of modified memory was varied from 5 Bytes to 10 % of 128 KB. The detection rate is measured for 10 iterations. The graphical representation of modified memory detection rate IRO all the variants of OWCAP is shown in figure-4.6 and detailed results are in successive paras.

4.6.2.1 Memory Detection Rate of OWCAP Using RC-4

Table 4.3 Memory Detection Rate Using RC-4

Ser	Memory Size	Modified Memory	No of Iterations	Memory Detection Percentage
1.	128 KB	10 %	10	100 %
2.	128 KB	5 %	10	100 %
3.	128 KB	1 %	10	100 %
4.	128 KB	0.9 %	10	100 %

5.	128 KB	0.8 %	10	100 %
6.	128 KB	0.5 %	10	90 %
7.	128 KB	0.1 %	10	30 %
8.	128 KB	60 Bytes	10	30 %
9.	128 KB	50 Bytes	10	30 %
10.	128 KB	40 Bytes	10	30 %
11.	128 KB	30 Bytes	10	10 %
12.	128 KB	20 Bytes	10	10 %
13.	128 KB	10 Bytes	10	10 %
14.	128 KB	8 Bytes	10	0 %
15.	128 KB	5 Bytes	10	0 %

4.6.2.2Memory Detection Rate of OWCAP Using RC-5

Table 4.4 Memory Detection Rate Using RC-5

Ser	Memory Size	Modified Memory	No of Iterations	Memory Detection Percentage
1.	128 KB	10 %	10	100%
2.	128 KB	5 %	10	100 %
3.	128 KB	1 %	10	100 %
4.	128 KB	0.9 %	10	80 %
5.	128 KB	0.8 %	10	80 %
6.	128 KB	0.5 %	10	40 %
7.	128 KB	0.1 %	10	40 %
8.	128 KB	60 Bytes	10	40 %
9.	128 KB	50 Bytes	10	40 %

10.	128 KB	40 Bytes	10	40 %
11.	128 KB	30 Bytes	10	40 %
12.	128 KB	20 Bytes	10	30 %
13.	128 KB	10 Bytes	10	0 %
14.	128 KB	8 Bytes	10	0 %
15.	128 KB	5 Bytes	10	0 %

4.6.2.3 Memory Detection Rate of OWCAP Using Skipjack

Table 4.5 Memory Detection Rate Using Skipjack

Ser	Memory Size	Modified Memory	No of Iterations	Memory Detection Percentage
1.	128 KB	10 %	10	100 %
2.	128 KB	5 %	10	100 %
3.	128 KB	1 %	10	100 %
4.	128 KB	0.9 %	10	80 %
5.	128 KB	0.8 %	10	80 %
6.	128 KB	0.5 %	10	50 %
7.	128 KB	0.1 %	10	40 %
8.	128 KB	60 Bytes	10	40 %
9.	128 KB	50 Bytes	10	30 %
10.	128 KB	40 Bytes	10	10 %
11.	128 KB	30 Bytes	10	0 %
12.	128 KB	20 Bytes	10	0 %
13.	128 KB	10 Bytes	10	0 %
14.	128 KB	8 Bytes	10	0 %

15.	128 KB	5 Bytes	10	0 %

4.6.2.4 Memory Detection Rate of OWCAP Using SHA-2

Table 4.6 Memory Detection Rate Using SHA-2

Ser	**Memory Size**	**Modified Memory**	**No of Iterations**	**Memory Detection Percentage**
1.	128 KB	10 %	10	100 %
2.	128 KB	5 %	10	100 %
3.	128 KB	1 %	10	100 %
4.	128 KB	0.9 %	10	20 %
5.	128 KB	0.8 %	10	10 %
6.	128 KB	0.5 %	10	10 %
7.	128 KB	0.1 %	10	10%
8.	128 KB	60 Bytes	10	0 %
9.	128 KB	50 Bytes	10	0 %
10.	128 KB	40 Bytes	10	0 %
11.	128 KB	30 Bytes	10	0 %
12.	128 KB	20 Bytes	10	0 %
13.	128 KB	10 Bytes	10	0 %
14.	128 KB	8 Bytes	10	0 %
15.	128 KB	5 Bytes	10	0 %

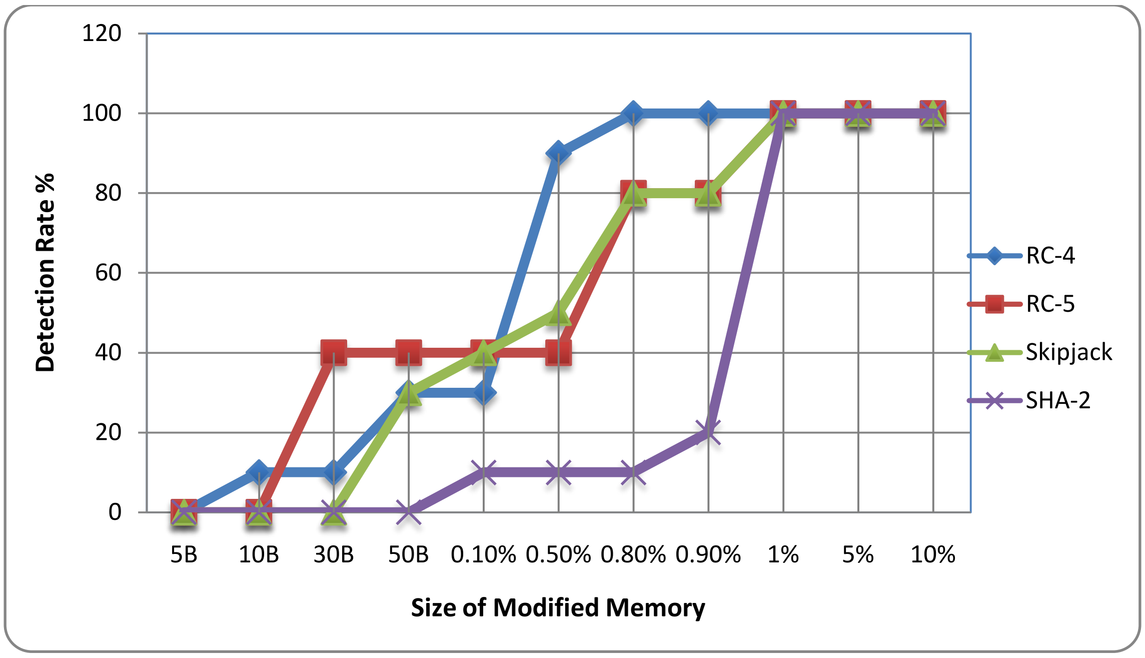


Figure-4.6 Detection rate of modified memory

4.7 Efficiency of OWCAP

The comparison of execution time and the modified memory detection rate (figure-4.6) shows that OWCAP is an efficient and a secure code attestation scheme that detects a code modification even if 0.8% of memory is modified as compare to OMAP which detects a 20 % modified memory with 95% probability. The execution time of OWCAP is also reduced to 2.1 m sec from 1.7 Sec of OMAP. If 1% memory modification is set as a base line and efficiency is also required then OWCAP SHA-2 version without encryption is the best choice otherwise if more emphasis is on security then OWCAP version of RC-4 is best suited since it detects a 0.8% memory modification with 100% probability, and if a more moderate solution is required that is economical as well as secure, then OWCAP version of RC-5 is recommended.

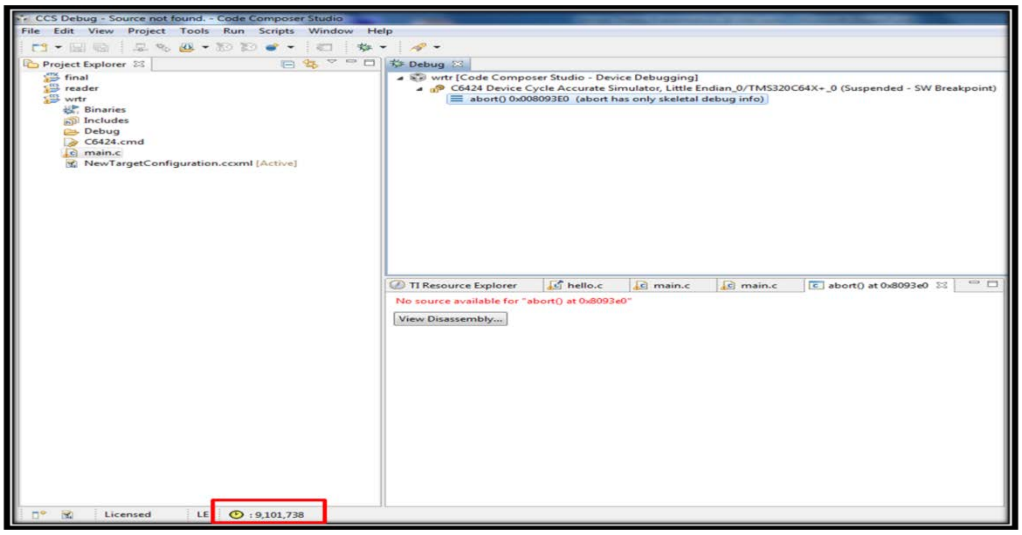
But here the question arises of energy efficiency of the proposed protocol. Therefore, in order to compute energy consumption of code attestation protocol, the RC-5 version of OWCAP is implemented in Code Composer Studio, which is the simulation / emulation software for Texas Instruments' microcontrollers. The simulation parameters are mentioned in Table 4.7 and the source code of OWCAP is mentioned at Appendix-K.

Table-4.7 Simulation Parameters

1.	Simulation Software	Code Composer version 5.2.1.00018
2.	Code Attestation Application Size	19 KB
3.	Microcontroller Family Simulated	TMS320C6424
4.	Features of TMS320C6424	
	a. RAM	240 KB
	b. Flash Memory	64 KB
	c. CPU Frequency	700/600/500/400 MHz
	d. Operating Voltage	1.2 V
	e. Current Consumption	597 mA per cycle for700 MHz processor

4.7.1 Energy Consumption of OWCAP

Energy consumption has been calculated based upon parameters mentioned in Table-4.7, of a single execution of OWCAP for TMS320C6424 family of microcontrollers, in Code Composer Studio. Once the OWCAP application is compiled in Code Composer with target device set as TMS320C6424, the code is compiled and builds as per the target microcontroller environment. Before the application code is built, the no of CPU Cycles can be measured by enabling the clock parameter before the code is run / debugged. The no of CPU cycles taken to build the OWCAP application are 91,01,738 (Figure-4.7).



```
wrtr [Code Composer Studio - Device Debugging]
  C6424 Device Cycle Accurate Simulator, Little Endian_0/TMS320C64X+_0 (Suspended - SW Breakpoint)
    abort() 0x008093E0  (abort has only skeletal debug info)

No source available for "abort() at 0x8093e0"

: 9,101,738
```

Figure-4.7 Number of CPU Cycles for OWCAP Execution

The clock frequency of target microcontroller is 700 MHz, therefore the execution time of OWCAP is CPU Cycles divided by Clock Frequency, which comes to 13 msec. The operating voltage of 700 MHz microcontroller is 1.2 V and current consumption is 597 mA per sec, hence the power consumption of device is = VI = 0.7164 Watt per second. Similarly the power consumption of the device for 13 msec is = 9.31 mWatt. The energy in Joules can be defined as the amount of power consumed for a particular period of time. In case of OWCAP the energy consumption is E_{CT} = 9.31 mWatt x 13 msec = 0.121mJ. Correspondingly the execution time and energy consumption of other microprocessors of the same family with different processing speed are also measured for single execution of OWCAP in Code Composer Studio; Details are mentioned in Table-4.8.

Table-4.8 Time and Energy Consumption of TMS320C6424 Family

TMS320C6424 Processor Speed (MHz)	Execution Time (m sec)	Energy Consumption (m Joules)
700	13	0.121
600	15.16	0.164
500	18.20	0.237
400	22.75	0.370

The graphical representation of comparison of execution time and energy consumption in respect of four different processors of target device is shown in Figure-4.8 and Figure-4.9 respectively. Moreover the relationship between execution time and energy consumption is shown in Figure-4.10. It can be clearly seen that the device with high speed processor gives most efficient performance in terms of execution time and the energy required to complete a single execution of OWCAP protocol. The more the clock speed of the processor of microcontroller, the less time it will take to execute the OWCAP protocol at runtime and resultantly will consume less energy as compared to slow processor devices.

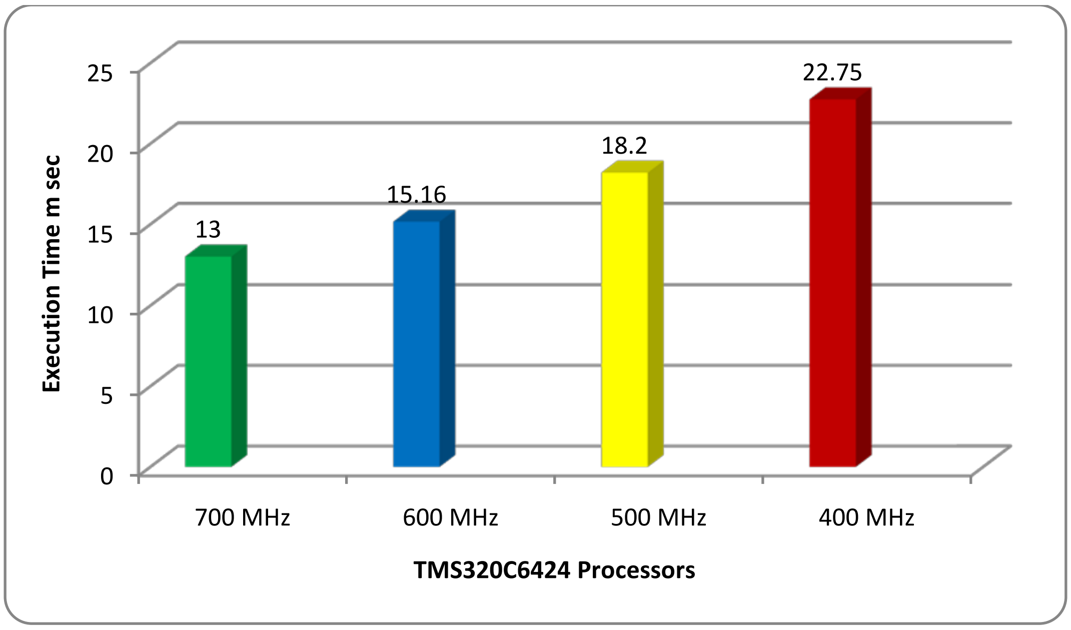


Figure-4.8 TMS320C6424 Processors' Execution Time

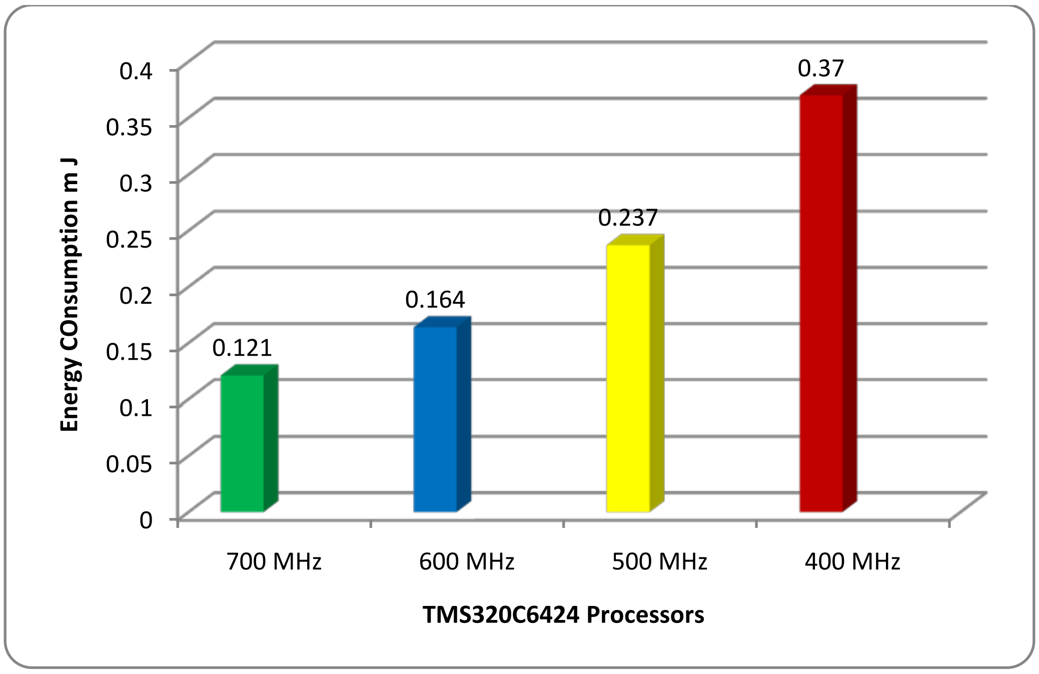


Figure-4.9 TMS320C6424 Processors' Energy Consumption

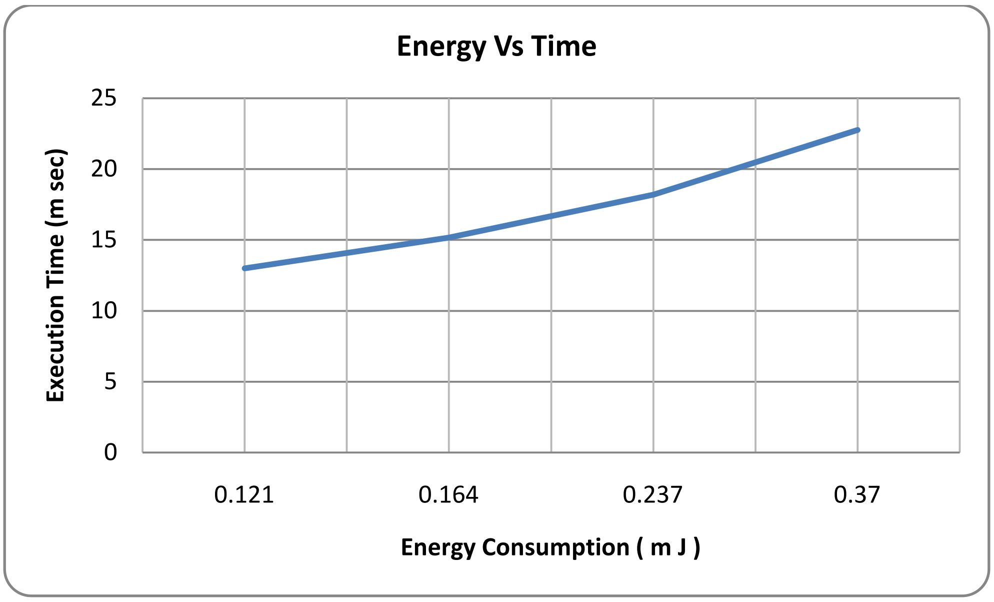


Figure-4.10 TMS320C6424 Processors' Energy Consumption Vs Execution Time

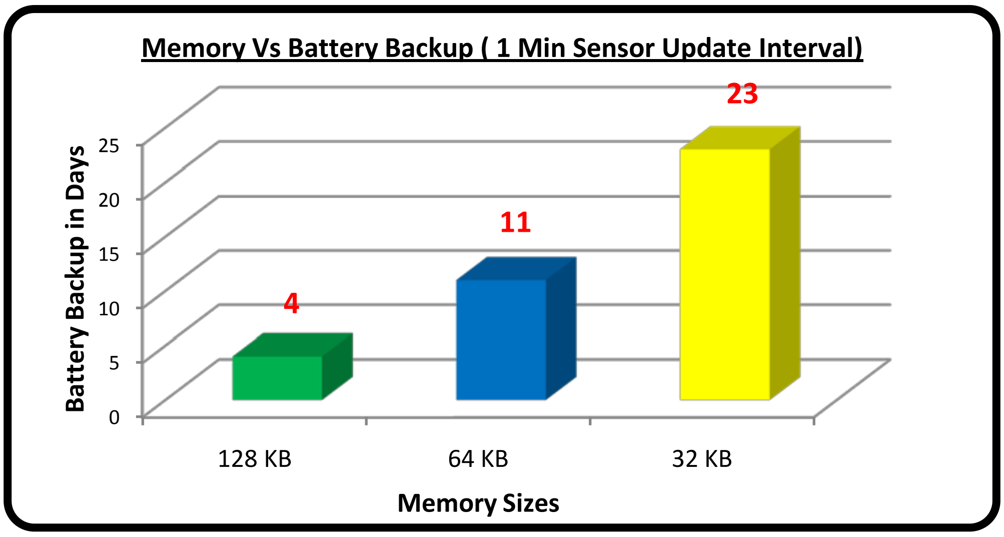


Figure 4.11 Memory Size Vs Battery Backup

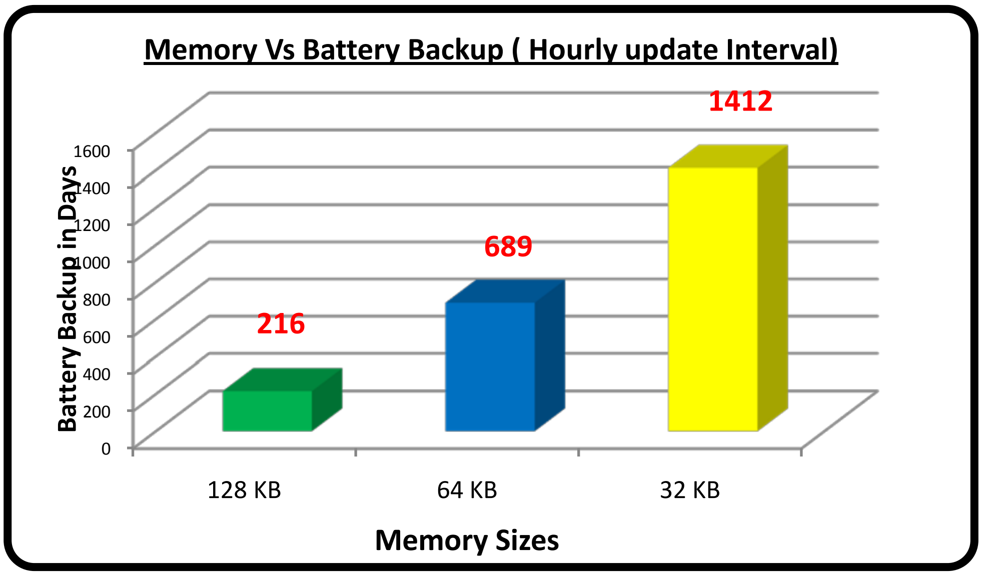


Figure 4.12 Memory Size Vs Battery Backup

4.7.2 OWCAP Battery Backup

Merely developing a security protocol is not enough. The security has to be evaluated again and again against other important parameters like performance, and efficiency. In case of wireless sensor networks and other SCADA networks the priorities are different, for a wireless sensor network, energy conservation is as important as security, because if a security protocol consumes a lot of energy, the battery will be drained in quick time, thus causing a hassle of replacing the battery of the sensor mote or remotely deploying a new mote. Considering a 700 MHz processor on board of a TMS320C6424 microcontroller the energy consumption for a single execution of OWCAP protocol is 0.121 m J. A typical battery installed on a wireless sensor has on average 2 J of on board energy. Hence there can be total 16529 executions of OWCAP protocol at runtime alongwith each sensor update. In an ICS environment,if a sensor update is deemed necessary at an hour interval, then in 24 hours there would be 24 executions of the proposed protocol. Cognizance to above rate the battery would require to be replaced after 1773 days in a star topology network, but in a mesh network the sensor node would require to relay messages from distant neighbours to the BS in this case the battery life will subsequently reduce. A comparison of battery life at different update periods is shown in Figure-4.11. The smaller the time between each sensor update the lesser is the battery life.

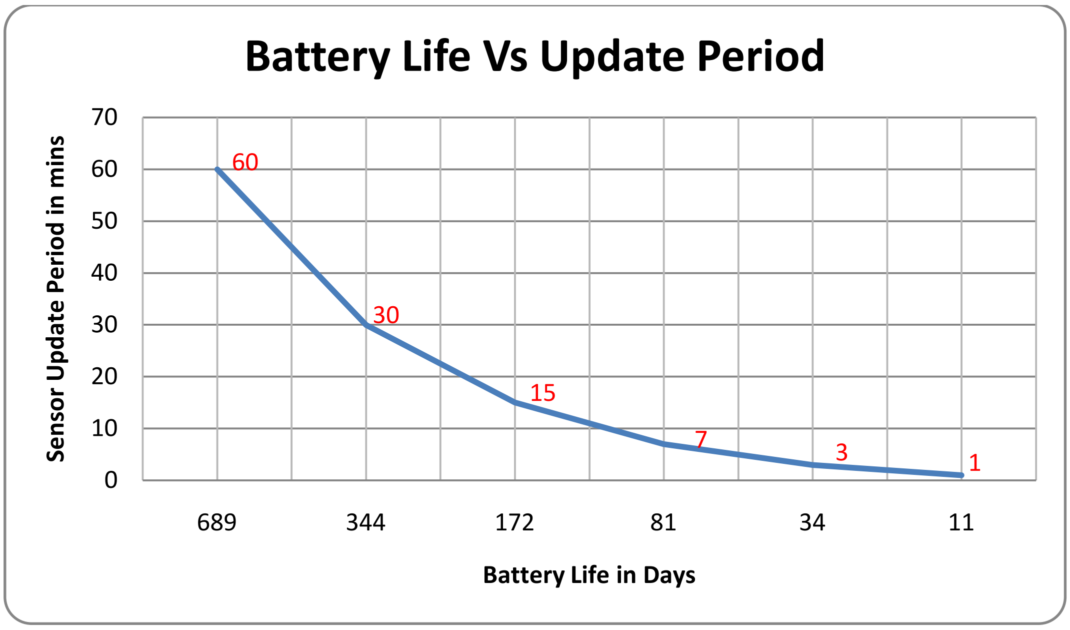


Figure-4.13 Battery Life Vs Update Period

4.8 Summary

OWCAP claims to provide maximum security for any ICS data network by performing code attestation of the sensor device at run time with each sensor update. Such a protocol is assumed to be computationally and power intensive. However the results achieved by OWCAP so far evidently show that with an intelligent selection of a microcontroller the energy consumption can be controlled, thus providing a longer battery backup time. Energy is considered to be a vital entity in case of wireless sensors, which have very limited resources in terms of on board processing power and battery backup. But similarly for a security hungry organization like nuclear power plants, smart grids, irrigation control system, and traffic control systems, requirement of security has an edge over other parameters; it is therefore a tradeoff between security and efficiency as per requirements of the organizations / ICS.

C h a p t e r 5

Comparative Analysis

5.1 Introduction

In order to determine that OWCAP is an economical and a secure scheme as compared to all other code attestation schemes discussed in chapter 3, a detailed and a unique comparative analysis of all the schemes is done. The comparison is based on parameters extracted from sensors' data sheet and the algorithms mentioned in all the schemes. Some details concerning our scenario for comparative analysis are shown in Table-5.1.

Table-5.1 WSN scenario for comparative analysis

Type of Mote	Mica2
Size of Program Flash Memory	128 Kbytes
Size of Code Test Application Binary	12 Kbytes
Nodes of Neighborhood	9 (Including BS)
Size of Hash Output	8 Bytes
Size of secret share	8 Bytes
Size of Challenge and Response	8 Bytes each
Symmetric Key	8 Bytes

5.2 Computational Cost

It is calculated for a single challenge response procedure in respect of all the schemes discussed before. In case of SWATT, attested node and verifier access memory for 1502.253K times and there are 1505.253K calls to RC-4 Cipher. Whereas in Basic Threshold Secret Sharing Scheme (BTSS), each node generates 116 KB of noise and make 116 K/ 8 = 15 K calls to RC-5cipher during noise generation. Each node does a (5-1) degree polynomial interpolation and evaluation. Each node computes the hash of the secret share. The verifier computes hash of recovered seed and noise for 116 KB memory size. Attested Node and verifier traverse memory for 376.313 K times and RC-5 cipher is called 376.313 K times for memory traversal. Considering Majority Voting Based Scheme (MVBS), each node generates 116 KB of noise. Attested node computes 8 responses to challenges sent by its 8 neighbors and access memory for 1505.253 K / 32 = 47 K times. Talking about OMAP, attested node and verifier make 1505.253 K/ 6 = 250.87 K accesses to memory for 20 percent memory modification. RC-4 cipher is called 1505.253 K/ 6 = 250.87K times. Attested node and verifier perform 2 hash computations each.Finally in case of proposed OWCAP, attested node and verifier make 376.313K accesses to memory in which RC-5 cipher is called 376.313K times. Attested node and verifier perform 1 hash and 1 MAC computation each.

5.3 Number of Messages Transmitted

It is also calculated for a single challenge response procedure. In case of SWATT, there is a challenge message from verifier to attested node and a response message from attested node to the verifier. Whereas in Basic Threshold Secret Sharing Scheme the node discovery and key establishment by all 9 nodes result into: 9 hello + 72reply + 72 key establishment + 72 ack = 297 Messages. Secret share distribution by each node requires72 messages. The cluster head nomination by 8 neighboring nodes requires8 messages. Cluster Head issues the challenge message to attested node. For secret Share collection minimum 5 messages (in case threshold for secret share is 5) are required and finally the attested node send response message to cluster head. The

Majority Voting Based Scheme require 72 messages for distributing challenge-response tuples, 8 challenge messages, 8 response messages and 8 vote messages. In case of OMAP we need to send a query message from verifier to attested node and a checksum response message from attested node to the verifier. Finally our Proposed OWCAP requires a sensor reading update message with implicit MAC for code attestation and an ack message from the verifier to the attested node in case of a successful verification of MAC. A graphical comparison of number of messages transmitted in each scheme is as shown in figure 5.1.

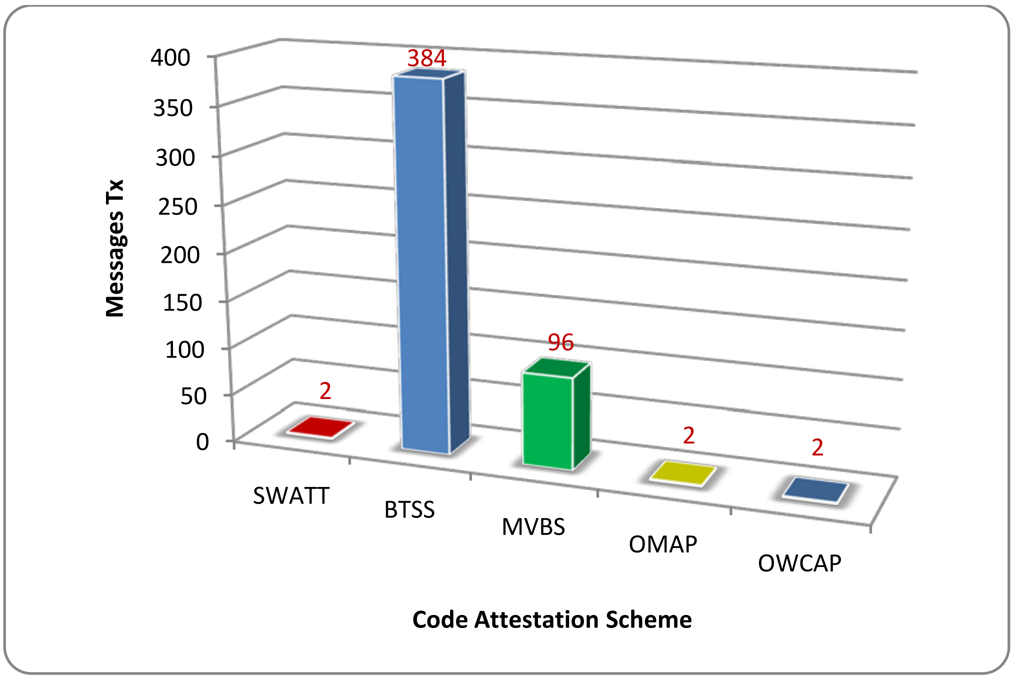


Figure 5.1 Comparison of Message Transmission

5.4 Storage Requirement

It is also calculated for a single challenge response procedure. In SWATT, verifier has to store memory images of 8 nodes i.e 8*128K = 1024K Byte. Whereas in Basic Threshold Secret Sharing Scheme, each node stores, pairwise keys with its neighbours,8 secret shares and 8 hashes. As the role of cluster-head is rotated for every attestation so each node stores memory image of 8 nodes i.e 8*128K = 1024K Bytes. In case of Majority Voting Based Scheme, each node stores 8 challenge-response pairs but the verifier does not need to store the memory image of the attested node as it does not locally computes the checksum at the verifier end. In OMAP, verifier stores memory image of 8 nodes i.e 8*128K = 1024 K Bytes. OMAP does not talk about any security measure related to confidentiality of the message or the integrity of the sender. In case of OWCAP, verifier (BS) stores memory image of 8 nodes i.e 8*128K = 1024K Bytes and also the key files containing secret keys for computing MAC in respect of 8 nodes linked with it (64 Bytes). Each node stores the key file containing secret keys shared with the verifier (BS) for computing HMAC (8 Bytes).

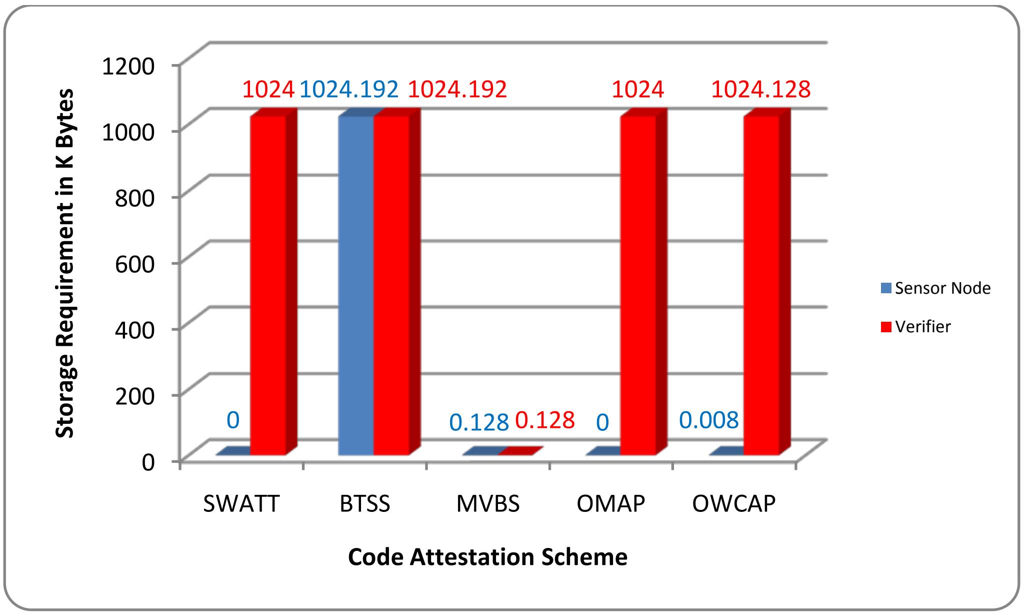


Figure-5.2 Storage Requirement of Code Attestation Schemes

5.5 Code Attestation Initiated by Whom and When

In SWATT, the verifier initiates the code attestation procedure on suspicion of abnormal behavior by an end node. In Basic Threshold Secret Sharing Scheme, if more than half neighbors detect abnormal behavior by an end node, then code attestation procedure is initiated. In Majority Voting Based Scheme also, If more than half neighbors detect abnormal behavior then code attestation procedure is initiated. Whereas in OMAP it is not mentioned clearly but verifier may send a query to attested node asking for the memory checksum. In case of our proposed OWCAP no explicit initiation is required, as the code attestation is implicit to each sensor reading update.

5.6 Security Aspects

Only OWCAP provides implicit code attestation integrated with routine application running on a sensor node. All other schemes perform code attestation as a separate application. OWCAP also diminishes the TOCTTOU gap by providing sensor update and the code attestation at the same time. Whereas all other schemes are vulnerable to attacks between TOCTTOU. Proposed OWCAP provides message integrity and message authentication using RC-5 cipher, already implemented in Tinysec [11] a security protocol for TinyOS-1.x. Whereas, SWATT [22] does not discuss about any such protection. The Basic Threshold Secret Sharing Scheme and Majority Voting Based Scheme [23], provide message confidentiality and message authentication but they do not specifically talk about integrity of challenge-response messages. Majority Voting Based Scheme has a drastic vulnerability in which the verifier does compute the memory checksum for the attested node and just compares the response received from the attested node with the expected response, hence any attacker compromising a legitimate node can send fake challenge response tuples and get verified / attested. OMAP also does not discuss at all about any such message protection measure. Being a challenge-response based scheme SWATT, Basic Threshold Secret Sharing and Majority Voting Based Schemes are vulnerable to network attacks like rainbow, interference and message fabrication attacks. The Basic Threshold Secret Sharing Scheme is also vulnerable to message modifications by compromised nodes and MITM (Man In The Middle) attacks during secret share collection procedure. Such a modification results into increased energy consumption and ultimately a DoS attack. Due to non availability of any message protection mechanism, OMAP is vulnerable to message modification and message forging attacks. Proposed OWCAP avoids network attacks like rainbow and interference attacks due to one way code attestation procedure and it also protects against message integrity attacks like message forging and message alteration. Unlike other schemes, OWCAP does provide a sensor node revocation and recovery procedure.

5.7 Bandwidth (BW) Utilization

It is based upon the number of messages transmitted for a single challenge-response procedure for code attestation scheme. Along with SWATT and OMAP, OWCAP has low bandwidth utilization as compared to Basic Threshold Secret Sharing Scheme and Majority Voting Based Scheme. Suppose each message transmission requires 3 ms of air time, then the comparative BW utilization for a single challenge-response protocol is as shown in figure 5.3.

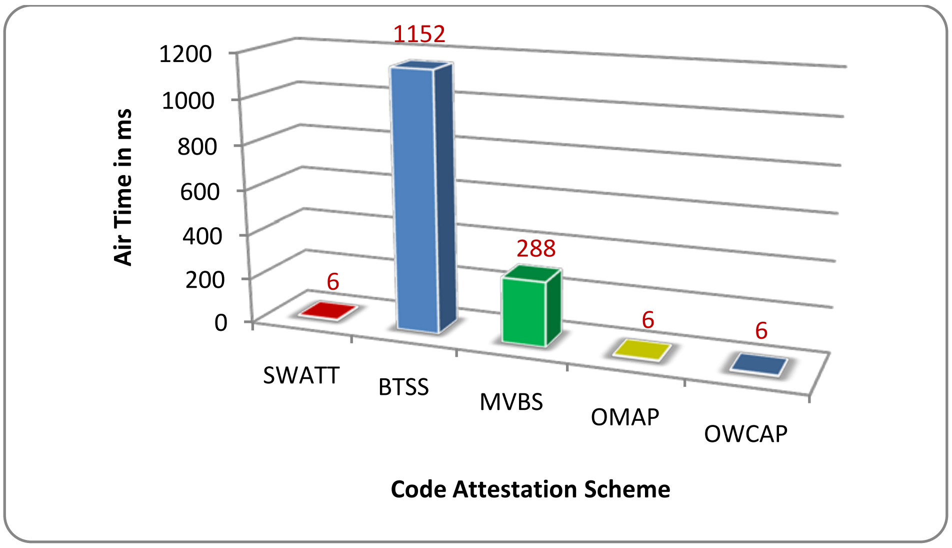


Figure-5.3 BW Utilization of Code Attestation Schemes

5.8 Summary

Being a one way code attestation protocol, OWCAP avoids network attacks like Rainbow and Interference Attacks. It provides message authentication and integrity with optional confidentiality. Most importantly it avoids attacks within TOCTTOU gap by integrating code attestation with routine sensor updates, thus providing an ideal solution for the networks demanding high level of security with an acceptable compromise on computational cost for every sensor update. As OWCAP performs code attestation at the time of sensor update, it successfully avoids a root kit based return oriented programming attack [27], which all previously discussed code attestation schemes fail to protect against. Some of the attacks on challenge response protocols and OMAP are shown in figure-5.4.

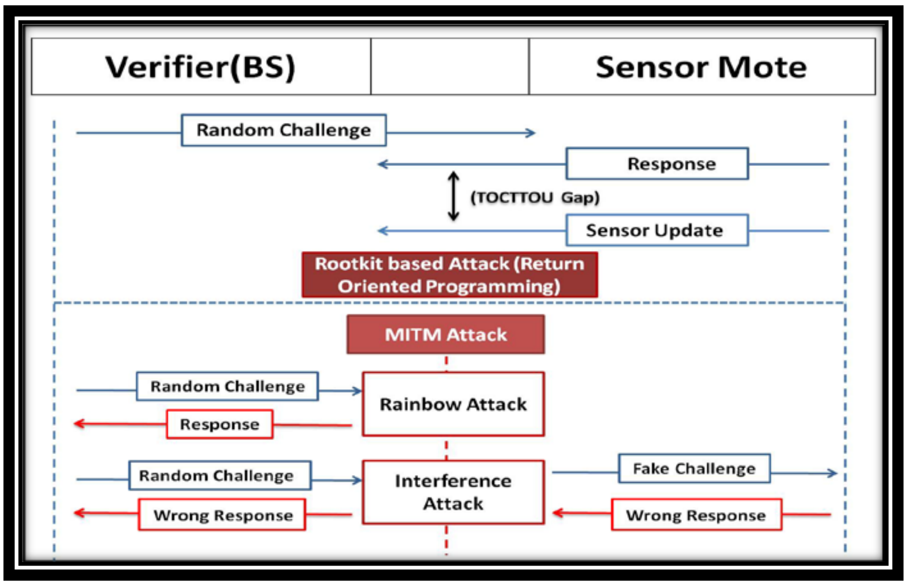


Figure-5.4Attacks on Challenge Response Protocols

Although OWCAP seems to be computationally more intensive than OMAP, yet it saves upon the precious energy by cutting down the need to transmit separate messages for code attestation and sensor updates. The use of secret serial number stored in an un-forgeable area of memory, binds the hardware to the verification procedure, so it is not possible for the attacker to change the hardware and send a correct memory checksum

[24]. The scheme also provides a much needed, node recovery and revocation procedure, which is initiated for a node with modified code upon reaching a certain threshold, thus reducing the probability of false initiation of recovery procedure and also reduce the probability of false revocation.

Chapter 6

Conclusion and Future Work

6.1 Introduction

In this chapter, the thesis has been concluded. Some possible enhancements of the work have been given in Section 6.3.

6.2 Conclusion

OWCAP is a technique that provides implicit protection against not only Sybil Attack but also other insider attacks that involve a sensor node with a malicious code. Based upon the efficient and secure performance of OWCAP discussed in Chapter 4 and comparative analysis explicated in chapter 5 it is establishedthat OWCAP provides maximum security by keeping the computation, transmission and storage overheads to a minimum level. It therefore substantiates the claim that OWCAP is an economical code attestation scheme with no compromise on security.

6.3 Future Work

At the moment OWCAP is restricted to star network topology in which only a cluster head has the responsibility of attestation and verifying a sensor node. The same scheme can be optimized for use in a heterogeneous network where all the neighboring nodes should be able to attest and verify its respective neighbors.The implementation of OWCAP on physical sensor motes will certainly give more precise results and help in pragmatic performance analysis.

6.4 Summary

The contribution of the thesis is the development of a novel code attestation scheme that is to be used not only in sensor networks but is applicable to any type of

ICS,where integrity of data and the sender is of utmost importance. Such secure sensing systems can be used for battle field surveillance, industrial control systems, process control / automation, traffic management systems, environmental monitoring, smart grids, nuclear power plants and smart houses.A one way code attestation was considered to be a computationally and energy intensive protocol but the proposed OWCAP has proved that for security starved organizations, a runtime code attestation is very much possible. It not only protects the critical / sensitive infrastructure from any malicious cyber attack involving code modification, but also prevent the control systems from ultimate failure.

APPENDICES

Appendix-A

DEV C++ CODE OF OWCAP (RC-4) FOR SENSOR NODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <openssl/sha.h>
#include <openssl/rc4.h>
#include <openssl/bio.h>
#include <openssl/hmac.h>
#include <conio.h>
#include <windows.h>
long GetTimeMs64()
{
 FILETIME ft;
 LARGE_INTEGER li;

 /* Get the amount of 100 nano seconds intervals elapsed since January
1, 1601 (UTC) and copy it
  * to a LARGE_INTEGER structure. */
 GetSystemTimeAsFileTime(&ft);
 li.LowPart = ft.dwLowDateTime;
 li.HighPart = ft.dwHighDateTime;

 long ret = li.QuadPart;
 ret -= 116444736000000000LL;

 return ret;

}
main( )
{
    long t0 = GetTimeMs64();
   time_t now;
    struct tm * ptm;
    now = time(NULL);
    ptm = gmtime ( &now );
    char t_stamp[10];
    char mon1[2];
    char day[2];
    char hour[2];
    char minutes[2];
    char seconds[2];
    if((ptm->tm_mon+1)<10)
    {
        sprintf(t_stamp,"0%d",ptm->tm_mon+1);
    }
    else
    {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

    if(ptm->tm_mday<10)
       { sprintf(day,"0%d",ptm->tm_mday);}
    else
        {sprintf(day,"%d",ptm->tm_mday);}
        strcat(t_stamp,day);
  if((ptm->tm_hour+5)<10)
      {sprintf(hour,"0%d",ptm->tm_hour+5);}
  else
      {sprintf(hour,"%d",ptm->tm_hour+5);}
      strcat(t_stamp,hour);
   if(ptm->tm_min<10)
      {sprintf(minutes,"0%d",ptm->tm_min);}
  else
      {sprintf(minutes,"%d",ptm->tm_min);}
      strcat(t_stamp,minutes);
   if(ptm->tm_sec<10)
      {sprintf(seconds,"0%d",ptm->tm_sec);}
  else
      {sprintf(seconds,"%d",ptm->tm_sec);}
      strcat(t_stamp,seconds);
    printf("\nt_stamp: %s\n",t_stamp);
  unsigned long time_stamp=atoi(t_stamp);
  unsigned int sensor_reading=ptm->tm_sec%10+1;
  char s_reading[2];
  sprintf(s_reading,"%d",sensor_reading);

  printf("time: %d\n",time_stamp);

clock_t begin, end;
clock_t begin_sha,end_sha;
clock_t begin_rc5,end_rc5;
clock_t begin_file,end_file;
clock_t begin_hmac,end_hmac;
double time_spent;
double time_spent_sha=0;
double time_spent_rc5=0;
double time_spent_file=0;
double time_spent_hmac=0;
  begin = clock();
  int j,i,y,k,l,m,decimal;
  int index = 0;
  char content[400] = {0x00};

  FILE *fp;
   char * buffer = 0;
   long length;
   FILE * f = fopen ("memread.txt", "rb");

   if (f)
   {
     fseek (f, 0, SEEK_END);
     length = ftell (f);
     fseek (f, 0, SEEK_SET);
     buffer = malloc (length);
     if (buffer)
     {
        fread (buffer, 1, length, f);
     }
     fclose (f);
   }
  unsigned char ibuf[32];
  unsigned char obuf[64];
  char hashString[64];
  char cipherString[64];
  int hexadecimal;
```

```
      int file_counter=0;
     int srno= 65535;
     int loop_saver;
     for(i = 0; i<20; i++)
     {
         begin_sha=clock();
         long combo = i + srno+time_stamp;
         sprintf(ibuf, "%d", combo);
         SHA256(ibuf,strlen(ibuf),obuf);
         for (j=0;j<32;j++)
         { sprintf (&hashString[j*2],"%02x", (unsigned int)obuf[j]);

         }

         end_sha=clock();
         time_spent_sha+= (double)(end_sha - begin_sha);
         int i;

        RC4_KEY key;

         static unsigned char key_data[64] =
{0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef};
         char* data_to_encrypt = hashString;
            unsigned char ciphertext[64];


         RC4_set_key(&key, 8, key_data);
         RC4(&key,strlen(data_to_encrypt), data_to_encrypt, ciphertext);
           end_rc5=clock();
           time_spent_rc5+= (double)(end_rc5 - begin_rc5);
                     for (l=0;l<32;l++)
           {
             loop_saver=l;
               for (m=0;m<3;m++)
               {
                     sprintf (&cipherString[m*2],"%02x", (unsigned
int)ciphertext[l]);               if (m ==2)
                          {
                              cipherString[5]=0;

                              l=loop_saver;
                              break;
                          }
                     l=l+1;
                }
                y = (int) strlen(cipherString);
                decimal = 0;
                k=0;
                int ch;
                while(y-1 != 0)
                {
                     ch = cipherString[k];
                     if('0' <= ch && ch <= '9')
                     {
                          decimal = decimal * 16;
                          decimal = decimal + (ch - '0');
                     }
                     else if('A' <= ch && ch <= 'F')
                     {
                        decimal = decimal * 16;
                        decimal = decimal + (ch - 'A')+10;
```

```
                    }
                    else if('a' <= ch && ch <= 'f')
                    {
                         decimal = decimal * 16;
                         decimal = decimal + (ch - 'a')+10;
                    }
                    else
                    {
                         decimal=0;
                         break;
                    }
                    y--;
                    k++;
               }

               int found = 0;
               char chartest[300] = {0x00};
               if(decimal<125000)
               {
                    chartest[0] = buffer[decimal];
                    if (isalnum(*chartest) || *chartest == '_')
                    {
                      content[index++] = buffer[decimal];
          }
               }

               if(found == 0)
                 printf("");
          }
              }
     int h;
      begin_hmac=clock();
     char key2[] = "012345678";
     unsigned char* digest;
     char mdString[40];
     sprintf(content,"%s%d",content,sensor_reading);

     digest = HMAC(EVP_sha1(), key2, strlen(key2), (unsigned
char*)content, strlen(content), NULL, NULL);


     for(h = 0; h < 20; h++)
     {
           sprintf (&mdString[h*2], "%02x", (unsigned int)digest[h]);
     }
     printf("\nmdString: %s,t_stamp: %s,sensor_reading:
%s",mdString,t_stamp,s_reading);

      end_hmac=clock();
      time_spent_hmac+= (double)(end_hmac - begin_hmac);
     // hmac ends here
      end = clock();
      time_spent = (double)(end - begin);
     double exectime = time_spent/CLOCKS_PER_SEC;
     long t1 = GetTimeMs64();
     char * f_result=0;
      printf("\nThe Execution Time in nano seconds is %d ",t1-t0);
     FILE *fh = fopen("out.dat", "w");
     fprintf(fh,"%s,%s,%s\n",mdString,t_stamp,s_reading);
     close(fh);
     getch(); }
```

Appendix-B

DEV C++ CODE OF OWCAP (RC-4) FOR VERIFIER

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <openssl/sha.h>
#include <openssl/rc4.h>
#include <openssl/bio.h>
#include <openssl/hmac.h>
#include <conio.h>
#include <windows.h>
long GetTimeMs64()
{
 FILETIME ft;
 LARGE_INTEGER li;

 /* Get the amount of 100 nano seconds intervals elapsed since January
1, 1601 (UTC) and copy it
  * to a LARGE_INTEGER structure. */
 GetSystemTimeAsFileTime(&ft);
 li.LowPart = ft.dwLowDateTime;
 li.HighPart = ft.dwHighDateTime;

 long ret = li.QuadPart;
 ret -= 116444736000000000LL; /* Convert from file time to UNIX epoch
time. */
 //ret /= 10000; /* From 100 nano seconds (10^-7) to 1 millisecond
(10^-3) intervals */

 return ret;

}
main( )
{
/* Reading the output file: Receiving the packet from network node 1 */
     char * rx_data = 0;
     long length;
     FILE * fp = fopen ("out.dat", "rb");

     if (fp)
     {
       fseek (fp, 0, SEEK_END);
       length = ftell (fp);
       fseek (fp, 0, SEEK_SET);
       rx_data = malloc (length);
       if (rx_data)
       {
          fread (rx_data, 1, length, fp);
       }
       fclose (fp);
     }
     char * msg_digest;
     char * time_n1;
     char * snsr1_reading;
     int snsr1_reading_i;
     /* Segregating the information from the received data */
     msg_digest = strtok(rx_data, ",");
     time_n1 = strtok(NULL, ",");
      snsr1_reading = strtok(NULL, ",");

     snsr1_reading = strtok(snsr1_reading, "\n");

    printf("\nReceived data\nMsg Digest n1: %s", msg_digest);
    printf("\nTime n1:%s", time_n1);
     printf("\nSensor reading n1: %s\n", snsr1_reading);
     snsr1_reading_i=atoi(snsr1_reading);
    long t0 = GetTimeMs64();
    time_t now;
    struct tm * ptm;
    now = time(NULL);
    ptm = gmtime ( &now );
    char t_stamp[10];

     char mon1[2];
    char day[2];
    char hour[2];
    char minutes[2];
    char seconds[2];
    if((ptm->tm_mon+1)<10)
    {
        sprintf(t_stamp,"0%d",ptm->tm_mon+1);
    }
    else
    {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

    if(ptm->tm_mday<10)
       { sprintf(day,"0%d",ptm->tm_mday);}
    else
        {sprintf(day,"%d",ptm->tm_mday);}
         strcat(t_stamp,day);
    if((ptm->tm_hour+5)<10)
        {sprintf(hour,"0%d",ptm->tm_hour+5);}
    else
        {sprintf(hour,"%d",ptm->tm_hour+5);}
        strcat(t_stamp,hour);
     if(ptm->tm_min<10)
        {sprintf(minutes,"0%d",ptm->tm_min);}
    else
        {sprintf(minutes,"%d",ptm->tm_min);}
        strcat(t_stamp,minutes);
     if(ptm->tm_sec<10)
        {sprintf(seconds,"0%d",ptm->tm_sec);}
    else
        {sprintf(seconds,"%d",ptm->tm_sec);}
        strcat(t_stamp,seconds);
    unsigned long time_stamp=atoi(t_stamp);
    unsigned long time_stamp_n1=atoi(time_n1);
    printf("current time:%d",time_stamp);
    int time_difference=time_stamp-time_stamp_n1;
    printf("\ntime difference: %d",time_difference);
    if(time_difference<10)
    {
         printf("\nTime difference is out of acceptable range. Packet
dropped!\nPress any key to continue..");
          getch();
          return 1;
    }
    else if(snsr1_reading_i<=0 || snsr1_reading_i>10)
    {
         printf("\nSensor range is out of acceptable range. Packet
dropped!\nPress any key to continue..");
          getch();
          return 1;
    }
    else
    {
          clock_t begin, end;
          clock_t begin_sha,end_sha;
          clock_t begin_rc5,end_rc5;
          clock_t begin_file,end_file;
          clock_t begin_hmac,end_hmac;
          double time_spent;
          double time_spent_sha=0;
          double time_spent_rc5=0;
          double time_spent_file=0;
          double time_spent_hmac=0;
          begin = clock();
          int j,i,y,k,l,m,decimal;
          int index = 0;
          char content[400] = {0x00};

          char * buffer = 0;
          FILE * f = fopen ("memread.txt", "rb");

          if (f)
          {
            fseek (f, 0, SEEK_END);
            length = ftell (f);
            fseek (f, 0, SEEK_SET);
            buffer = malloc (length);
            if (buffer)
            {
               fread (buffer, 1, length, f);
            }
            fclose (f);
          }
          unsigned char ibuf[32];
          unsigned char obuf[64];
          char hashString[64];
          char cipherString[64];
          int hexadecimal;
          int file_counter=0;
          int srno= 65535;
          int loop_saver;
          for(i = 0; i<20; i++)
          {
               begin_sha=clock();
               long combo = i + srno+time_stamp_n1;
               sprintf(ibuf, "%d", combo);
               SHA256(ibuf,strlen(ibuf),obuf);
               for (j=0;j<32;j++)
               { sprintf (&hashString[j*2],"%02x", (unsigned
int)obuf[j]);

               }

               end_sha=clock();
               time_spent_sha+= (double)(end_sha - begin_sha);
```

```
             int i;
         RC4_KEY key;

            static unsigned char key_data[64] =
                         {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef};
            char* data_to_encrypt = hashString;
            unsigned char ciphertext[64];

         RC4_set_key(&key, 8, key_data);
         RC4(&key,strlen(data_to_encrypt), data_to_encrypt, ciphertext);
          end_rc5=clock();
          time_spent_rc5+= (double)(end_rc5 - begin_rc5);
             for (l=0;l<32;l++)
             {
               loop_saver=l;
               for (m=0;m<3;m++)
               {
                   sprintf (&cipherString[m*2],"%02x", (unsigned
int)ciphertext[l]);
                if (m ==2)
                      {
                         cipherString[5]=0;

                         l=loop_saver;
                         break;
                      }
                   l=l+1;
                }
                y = (int) strlen(cipherString);
                decimal = 0;
                k=0;
                int ch;
                while(y-1 != 0)
                {
                   ch = cipherString[k];
                   if('0' <= ch && ch <= '9')
                   {
                      decimal = decimal * 16;
                      decimal = decimal + (ch - '0');
                   }
                   else if('A' <= ch && ch <= 'F')
                   {
                     decimal = decimal * 16;
                     decimal = decimal + (ch - 'A')+10;
                   }
                   else if('a' <= ch && ch <= 'f')
                   {
                      decimal = decimal * 16;
                      decimal = decimal + (ch - 'a')+10;
                   }
                   else
                   {
                      decimal=0;
                      break;
                   }
                   y--;
                   k++;
                }
                file_counter+=1;
                int found = 0;
                char chartest[300] = {0x00};
```

```
                    if(decimal<125000)
                    {
                         chartest[0] = buffer[decimal];
                         if (isalnum(*chartest) || *chartest == '_')
                         {
                           content[index++] = buffer[decimal];
                         }
                    }

                    if(found == 0)
                      printf("");
               }

          }
          int h;

          begin_hmac=clock();
          char key2[] = "012345678";
          unsigned char* digest;
          char mdString[40];

          sprintf(content,"%s%d",content,snsr1_reading_i);
          digest = HMAC(EVP_sha1(), key2, strlen(key2), (unsigned
char*)content, strlen(content), NULL, NULL);

          for(h = 0; h < 20; h++)
          {
               sprintf (&mdString[h*2], "%02x", (unsigned
int)digest[h]);
          }
        int ret=strncmp(mdString,msg_digest,40);
        if(ret==0)
          {
               printf("\n\nDigest match.\n");
          }
          else
          {
            printf("\n\nDigest does not match. Packet dropped.\nPress
any key to continue..\n");
        }
        printf("\nmdString: %s",mdString);

          end_hmac=clock();
          time_spent_hmac+= (double)(end_hmac - begin_hmac);
          // hmac ends here
          end = clock();
          time_spent = (double)(end - begin);
          double exectime = time_spent/CLOCKS_PER_SEC;
          long t1 = GetTimeMs64();
          char * f_result=0;

           printf("\nThe Execution Time in nano seconds is %d ",t1-
                 t0);

  }
    getche();
}
```

Appendix-C

DEV C++ CODE OF OWCAP (RC-5) FOR SENSOR NODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include<conio.h>
#include <openssl/sha.h>
#include <openssl/bio.h>
#include <openssl/hmac.h>
#include <windows.h>
typedef unsigned long int u32;
typedef unsigned char u16;

#define ROTL(x,y)  ( ((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))) )
#define ROTR(x,y)  ( ((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))) )
void RC5_encrypt( u32 *data );
void RC5_decrypt( u32 *data );
void key_setup( unsigned char *K );
#define w 32
#define r 1
#define b 16
#define c_no 4
#define t 26
u32 S[t];
u32 P = 0xb7e15163, Q = 0x9e3779b9;



void rc5_encrypt( u16 *data )
{
     u32 in[2]={0,0};
     u32 i,A,B;
     in[0] = ((u32)data[0]<<24) ^ ((u32)data[1]<<16) ^
((u32)data[2]<<8) ^ (u32)data[3];
     in[1] = ((u32)data[4]<<24) ^ ((u32)data[5]<<16) ^
((u32)data[6]<<8) ^ (u32)data[7];
     A=in[0]+S[0];
     B=in[1]+S[1];
     for( i=1; i<=r; i++ )
          {
               A = ROTL( A^B, B ) + S[2*i];
               B = ROTL( B^A, A ) + S[2*i+1];
          }
     in[0] = A;
     in[1] = B;

     data[0] = (u16)((in[0]>>24)  &0x000000ff);
     data[1] = (u16)((in[0]>>16)  &0x000000ff);
     data[2] = (u16)((in[0]>>8)   &0x000000ff);
     data[3] = (u16)(in[0]        &0x000000ff);
     data[4] = (u16)((in[1]>>24)  &0x000000ff);
     data[5] = (u16)((in[1]>>16)  &0x000000ff);
     data[6] = (u16)((in[1]>>8)   &0x000000ff);
     data[7] = (u16)(in[1]        &0x000000ff);

}
```

```
void rc5_decrypt( u16 *data )
{
     u32 in[2]={0,0};
     u32 i,B,A;
     in[0] = ((u32)data[0]<<24) ^ ((u32)data[1]<<16) ^
((u32)data[2]<<8) ^ (u32)data[3];
     in[1] = ((u32)data[4]<<24) ^ ((u32)data[5]<<16) ^
((u32)data[6]<<8) ^ (u32)data[7];
     B=in[1];
     A=in[0];
     for( i=r; i>0; i-- )
     {
          B = ROTR( B-S[2*i+1], A ) ^ A;
          A = ROTR( A-S[2*i], B ) ^ B;
     }
     in[1] = B - S[1];
     in[0] = A - S[0];

     data[0] = (u16)((in[0]>>24)  &0x000000ff);
     data[1] = (u16)((in[0]>>16)  &0x000000ff);
     data[2] = (u16)((in[0]>>8)   &0x000000ff);
     data[3] = (u16)(in[0]        &0x000000ff);
     data[4] = (u16)((in[1]>>24)  &0x000000ff);
     data[5] = (u16)((in[1]>>16)  &0x000000ff);
     data[6] = (u16)((in[1]>>8)   &0x000000ff);
     data[7] = (u16)(in[1]        &0x000000ff);
}

void rc5_key_setup( u16 *K )
{
     u32 i,j,k,u=w/8,A,B,L[c_no];
     for( i=b-1,L[c_no-1]=0; i!=-1; i-- )
     {
          L[i/u] = ( L[i/u]<<8 ) + K[i];
     }
     for( S[0]=P,i=1; i<t; i++ )
     {
          S[i] = S[i-1] + Q;
     }
     for( A=B=i=j=k=0; k<3*t; k++,i=(i+1)%t,j=(j+1)%c_no )
     {
          A = S[i] = ROTL( S[i]+(A+B), 3 );
          B = L[j] = ROTL( L[j]+(A+B), (A+B) );
     }
}
    double startTimeInMicroSec;
    double endTimeInMicroSec;
    int    stopped;
    LARGE_INTEGER frequency;
    LARGE_INTEGER startCount;
    LARGE_INTEGER endCount;
double getElapsedTime()
{
    if(!stopped)
        QueryPerformanceCounter(&endCount);
    startTimeInMicroSec = startCount.QuadPart * (1000000.0 /
frequency.QuadPart);
    endTimeInMicroSec = endCount.QuadPart * (1000000.0 /
frequency.QuadPart);
printf("\nValue of start in micro: %f\n",startTimeInMicroSec);
```

```
    return endTimeInMicroSec - startTimeInMicroSec;
}
#include <windows.h>
long GetTimeMs64()
{
 FILETIME ft;
 LARGE_INTEGER li;

 /* Get the amount of 100 nano seconds intervals elapsed since January
1, 1601 (UTC) and copy it
  * to a LARGE_INTEGER structure. */
 GetSystemTimeAsFileTime(&ft);
 li.LowPart = ft.dwLowDateTime;
 li.HighPart = ft.dwHighDateTime;

 long ret = li.QuadPart;
 ret -= 116444736000000000LL;

 return ret;

}
main( )
{
    long t0 = GetTimeMs64();
   time_t now;
    struct tm * ptm;
    now = time(NULL);
    ptm = gmtime ( &now );
    char t_stamp[10];
    char mon1[2];
    char day[2];
    char hour[2];
    char minutes[2];
    char seconds[2];
    if((ptm->tm_mon+1)<10)
    {
        sprintf(t_stamp,"0%d",ptm->tm_mon+1);
    }
    else
    {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

    if(ptm->tm_mday<10)
       { sprintf(day,"0%d",ptm->tm_mday);}
    else
        {sprintf(day,"%d",ptm->tm_mday);}
          strcat(t_stamp,day);
    if((ptm->tm_hour+5)<10)
        {sprintf(hour,"0%d",ptm->tm_hour+5);}
    else
        {sprintf(hour,"%d",ptm->tm_hour+5);}
        strcat(t_stamp,hour);
     if(ptm->tm_min<10)
        {sprintf(minutes,"0%d",ptm->tm_min);}
    else
        {sprintf(minutes,"%d",ptm->tm_min);}
        strcat(t_stamp,minutes);
     if(ptm->tm_sec<10)
        {sprintf(seconds,"0%d",ptm->tm_sec);}
    else
        {sprintf(seconds,"%d",ptm->tm_sec);}
        strcat(t_stamp,seconds);
      printf("\nt_stamp: %s\n",t_stamp);
    unsigned long time_stamp=atoi(t_stamp);
    unsigned int sensor_reading=ptm->tm_sec%10+1;
    char s_reading[2];
    sprintf(s_reading,"%d",sensor_reading);

    printf("time: %d\n",time_stamp);
  clock_t begin, end;
  clock_t begin_sha,end_sha;
  clock_t begin_rc5,end_rc5;
  clock_t begin_file,end_file;
  clock_t begin_hmac,end_hmac;
  double time_spent;
  double time_spent_sha=0;
  double time_spent_rc5=0;
  double time_spent_file=0;
  double time_spent_hmac=0;
    begin = clock();
    int j,i,y,k,l,m,decimal;
    int index = 0;
    char content[400] = {0x00};

    FILE *fp;
     char * buffer = 0;
     long length;
     FILE * f = fopen ("memread.txt", "rb");

     if (f)
     {
       fseek (f, 0, SEEK_END);
       length = ftell (f);
       fseek (f, 0, SEEK_SET);
       buffer = malloc (length);
       if (buffer)
       {
          fread (buffer, 1, length, f);
       }
       fclose (f);
     }

    unsigned char ibuf[32];
    unsigned char obuf[64];
    char hashString[64];
    char cipherString[64];
    int hexadecimal;
     int file_counter=0;
    int srno= 65535;
    int loop_saver;
    for(i = 0; i<20; i++)
    {
        begin_sha=clock();
        long combo = i + srno+time_stamp;
        sprintf(ibuf, "%d", combo);
        SHA256(ibuf,strlen(ibuf),obuf);
        for (j=0;j<32;j++)
        { sprintf (&hashString[j*2],"%02x", (unsigned int)obuf[j]);

        }

        end_sha=clock();
        time_spent_sha+= (double)(end_sha - begin_sha);
```

```
        int i;
    unsigned char key[b] =
          {
               0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80,
               0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
          };
          begin_rc5=clock();
          rc5_key_setup( key );
          rc5_encrypt( hashString );
          end_rc5=clock();
          time_spent_rc5+= (double)(end_rc5 - begin_rc5);
          for (l=0;l<sizeof(hashString);l++)
          {
           loop_saver=l;
            for (m=0;m<3;m++)
            {
                    sprintf (&cipherString[m*2],"%02x", (unsigned
int)hashString[l]);                if (m ==2)
                         {
                              cipherString[5]=0;

                              l=loop_saver;
                              break;
                         }
                    l=l+1;
               }
               y = (int) strlen(cipherString);
               decimal = 0;
               k=0;
               int ch;
               while(y-1 != 0)
               {
                    ch = cipherString[k];
                    if('0' <= ch && ch <= '9')
                    {
                         decimal = decimal * 16;
                         decimal = decimal + (ch - '0');
                    }
                    else if('A' <= ch && ch <= 'F')
                    {
                       decimal = decimal * 16;
                       decimal = decimal + (ch - 'A')+10;
                    }
                    else if('a' <= ch && ch <= 'f')
                    {
                         decimal = decimal * 16;
                         decimal = decimal + (ch - 'a')+10;
                    }
                    else
                    {
                         decimal=0;
                         break;
                    }
                    y--;
                    k++;
               }

               int found = 0;
               char chartest[300] = {0x00};
               if(decimal<125000)
               {
                    chartest[0] = buffer[decimal];
                    if (isalnum(*chartest) || *chartest == '_')
                    {
                      content[index++] = buffer[decimal];
          }
               }

               if(found == 0)
                 printf("");
          }

    }
    int h;
     begin_hmac=clock();
    char key2[] = "012345678";
    unsigned char* digest;
    char mdString[40];
    sprintf(content,"%s%d",content,sensor_reading);
    digest = HMAC(EVP_sha1(), key2, strlen(key2), (unsigned
char*)content, strlen(content), NULL, NULL);

    for(h = 0; h < 20; h++)
    {
          sprintf (&mdString[h*2], "%02x", (unsigned int)digest[h]);
    }
    printf("\nmdString: %s,t_stamp: %s,sensor_reading:
%s",mdString,t_stamp,s_reading);

     end_hmac=clock();
     time_spent_hmac+= (double)(end_hmac - begin_hmac);
    // hmac ends here
     end = clock();
     time_spent = (double)(end - begin);
    double exectime = time_spent/CLOCKS_PER_SEC;
    long t1 = GetTimeMs64();
    char * f_result=0;
    printf("\nThe Execution Time in nano seconds is %d ",t1-t0);
    FILE *fh = fopen("out.dat", "w");
    fprintf(fh,"%s,%s,%s\n",mdString,t_stamp,s_reading);
    close(fh);
    getch();
}
```

Appendix-D

DEV C++ CODE OF OWCAP (RC-5) FOR VERIFIER

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include<conio.h>
#include <openssl/sha.h>
#include <openssl/bio.h>
#include <openssl/hmac.h>
#include <windows.h>
typedef unsigned long int u32;
typedef unsigned char u16;
#define ROTL(x,y)  ( ((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))) )
#define ROTR(x,y)  ( ((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))) )
void RC5_encrypt( u32 *data );
void RC5_decrypt( u32 *data );
void key_setup( unsigned char *K );
#define w 32
#define r 1
#define b 16
#define c_no 4
#define t 26
u32 S[t];
u32 P = 0xb7e15163, Q = 0x9e3779b9;


void rc5_encrypt( u16 *data )
{
     u32 in[2]={0,0};
     u32 i,A,B;
     in[0] = ((u32)data[0]<<24) ^ ((u32)data[1]<<16) ^
((u32)data[2]<<8) ^ (u32)data[3];
     in[1] = ((u32)data[4]<<24) ^ ((u32)data[5]<<16) ^
((u32)data[6]<<8) ^ (u32)data[7];
     A=in[0]+S[0];
     B=in[1]+S[1];
     for( i=1; i<=r; i++ )
          {
               A = ROTL( A^B, B ) + S[2*i];
               B = ROTL( B^A, A ) + S[2*i+1];
          }
     in[0] = A;
     in[1] = B;

     data[0] = (u16)((in[0]>>24)  &0x000000ff);
     data[1] = (u16)((in[0]>>16)  &0x000000ff);
     data[2] = (u16)((in[0]>>8)   &0x000000ff);
     data[3] = (u16)(in[0]        &0x000000ff);
     data[4] = (u16)((in[1]>>24)  &0x000000ff);
     data[5] = (u16)((in[1]>>16)  &0x000000ff);
     data[6] = (u16)((in[1]>>8)   &0x000000ff);
     data[7] = (u16)(in[1]        &0x000000ff);

}

void rc5_decrypt( u16 *data )
```

```
{
     u32 in[2]={0,0};
     u32 i,B,A;
     in[0] = ((u32)data[0]<<24) ^ ((u32)data[1]<<16) ^
((u32)data[2]<<8) ^ (u32)data[3];
     in[1] = ((u32)data[4]<<24) ^ ((u32)data[5]<<16) ^
((u32)data[6]<<8) ^ (u32)data[7];
     B=in[1];
     A=in[0];
     for( i=r; i>0; i-- )
     {
          B = ROTR( B-S[2*i+1], A ) ^ A;
          A = ROTR( A-S[2*i], B ) ^ B;
     }
     in[1] = B - S[1];
     in[0] = A - S[0];

     data[0] = (u16)((in[0]>>24)  &0x000000ff);
     data[1] = (u16)((in[0]>>16)  &0x000000ff);
     data[2] = (u16)((in[0]>>8)   &0x000000ff);
     data[3] = (u16)(in[0]        &0x000000ff);
     data[4] = (u16)((in[1]>>24)  &0x000000ff);
     data[5] = (u16)((in[1]>>16)  &0x000000ff);
     data[6] = (u16)((in[1]>>8)   &0x000000ff);
     data[7] = (u16)(in[1]        &0x000000ff);
}

void rc5_key_setup( u16 *K )
{
     u32 i,j,k,u=w/8,A,B,L[c_no];
     for( i=b-1,L[c_no-1]=0; i!=-1; i-- )
     {
          L[i/u] = ( L[i/u]<<8 ) + K[i];
     }
     for( S[0]=P,i=1; i<t; i++ )
     {
          S[i] = S[i-1] + Q;
     }
     for( A=B=i=j=k=0; k<3*t; k++,i=(i+1)%t,j=(j+1)%c_no )
     {
          A = S[i] = ROTL( S[i]+(A+B), 3 );
          B = L[j] = ROTL( L[j]+(A+B), (A+B) );
     }
}
    double startTimeInMicroSec;
    double endTimeInMicroSec;
    int    stopped;
    LARGE_INTEGER frequency;
    LARGE_INTEGER startCount;
    LARGE_INTEGER endCount;

double getElapsedTime()
{
    if(!stopped)
        QueryPerformanceCounter(&endCount);

    startTimeInMicroSec = startCount.QuadPart * (1000000.0 /
frequency.QuadPart);
    endTimeInMicroSec = endCount.QuadPart * (1000000.0 /
frequency.QuadPart);
printf("\nValue of start in micro: %f\n",startTimeInMicroSec);
```

```
    return endTimeInMicroSec - startTimeInMicroSec;
}
#include <windows.h>

long GetTimeMs64()
{
 FILETIME ft;
 LARGE_INTEGER li;

 /* Get the amount of 100 nano seconds intervals elapsed since January
1, 1601 (UTC) and copy it
  * to a LARGE_INTEGER structure. */
 GetSystemTimeAsFileTime(&ft);
 li.LowPart = ft.dwLowDateTime;
 li.HighPart = ft.dwHighDateTime;

 long ret = li.QuadPart;
 ret -= 116444736000000000LL; /* Convert from file time to UNIX epoch
time. */
 //ret /= 10000; /* From 100 nano seconds (10^-7) to 1 millisecond
(10^-3) intervals */

 return ret;

}
main( )
{
/* Reading the output file: Receiving the packet from network node 1 */
     char * rx_data = 0;
     long length;
     FILE * fp = fopen ("out.dat", "rb");

     if (fp)
     {
       fseek (fp, 0, SEEK_END);
       length = ftell (fp);
       fseek (fp, 0, SEEK_SET);
       rx_data = malloc (length);
       if (rx_data)
       {
          fread (rx_data, 1, length, fp);
       }
       fclose (fp);
     }
     char * msg_digest;
     char * time_n1;
     char * snsr1_reading;
int snsr1_reading_i;
     /* Segregating the information from the received data */
     msg_digest = strtok(rx_data, ",");
     time_n1 = strtok(NULL, ",");
      snsr1_reading = strtok(NULL, ",");

     snsr1_reading = strtok(snsr1_reading, "\n");

    printf("\nReceived data\nMsg Digest n1: %s", msg_digest);
    printf("\nTime n1:%s", time_n1);
     printf("\nSensor reading n1: %s\n", snsr1_reading);
     snsr1_reading_i=atoi(snsr1_reading);
    long t0 = GetTimeMs64();
    time_t now;
    struct tm * ptm;
    now = time(NULL);
    ptm = gmtime ( &now );
    char t_stamp[10];

     char mon1[2];
    char day[2];
    char hour[2];
    char minutes[2];
    char seconds[2];
    if((ptm->tm_mon+1)<10)
    {
        sprintf(t_stamp,"0%d",ptm->tm_mon+1);
    }
    else
    {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

    if(ptm->tm_mday<10)
       { sprintf(day,"0%d",ptm->tm_mday);}
    else
        {sprintf(day,"%d",ptm->tm_mday);}
          strcat(t_stamp,day);
    if((ptm->tm_hour+5)<10)
        {sprintf(hour,"0%d",ptm->tm_hour+5);}
    else
        {sprintf(hour,"%d",ptm->tm_hour+5);}
        strcat(t_stamp,hour);
     if(ptm->tm_min<10)
        {sprintf(minutes,"0%d",ptm->tm_min);}
    else
        {sprintf(minutes,"%d",ptm->tm_min);}
        strcat(t_stamp,minutes);
     if(ptm->tm_sec<10)
        {sprintf(seconds,"0%d",ptm->tm_sec);}
    else
        {sprintf(seconds,"%d",ptm->tm_sec);}
        strcat(t_stamp,seconds);
    unsigned long time_stamp=atoi(t_stamp);
    unsigned long time_stamp_n1=atoi(time_n1);
    printf("current time:%d",time_stamp);
    int time_difference=time_stamp-time_stamp_n1;
    printf("\ntime difference: %d",time_difference);
    if(time_difference<10)
    {
         printf("\nTime difference is out of acceptable range. Packet
dropped!\nPress any key to continue..");
           getch();
           return 1;
    }
    else if(snsr1_reading_i<=0 || snsr1_reading_i>10)
    {
         printf("\nSensor range is out of acceptable range. Packet
dropped!\nPress any key to continue..");
           getch();
           return 1;
    }
    else
    {
          clock_t begin, end;
          clock_t begin_sha,end_sha;
          clock_t begin_rc5,end_rc5;
          clock_t begin_file,end_file;
          clock_t begin_hmac,end_hmac;
          double time_spent;
          double time_spent_sha=0;
          double time_spent_rc5=0;
          double time_spent_file=0;
          double time_spent_hmac=0;
          begin = clock();
          int j,i,y,k,l,m,decimal;
          int index = 0;
          char content[400] = {0x00};

          char * buffer = 0;
          FILE * f = fopen ("memread.txt", "rb");

          if (f)
          {
            fseek (f, 0, SEEK_END);
            length = ftell (f);
            fseek (f, 0, SEEK_SET);
            buffer = malloc (length);
            if (buffer)
            {
               fread (buffer, 1, length, f);
            }
            fclose (f);
          }
          unsigned char ibuf[32];
          unsigned char obuf[64];
          char hashString[64];
          char cipherString[64];
          int hexadecimal;
          int file_counter=0;
          int srno= 65535;
          int loop_saver;
          for(i = 0; i<20; i++)
          {
               begin_sha=clock();
               long combo = i + srno+time_stamp_n1;
               sprintf(ibuf, "%d", combo);
               SHA256(ibuf,strlen(ibuf),obuf);
               for (j=0;j<32;j++)
               { sprintf (&hashString[j*2],"%02x", (unsigned
int)obuf[j]);

               }

               end_sha=clock();
               time_spent_sha+= (double)(end_sha - begin_sha);
               int i;
               unsigned char key[b] =
          {
               0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80,
               0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
          };

          begin_rc5=clock();
          rc5_key_setup( key );
          rc5_encrypt( hashString );
          end_rc5=clock();
          time_spent_rc5+= (double)(end_rc5 - begin_rc5);
          for (l=0;l<sizeof(hashString);l++)
               {
                  loop_saver=l;
                  for (m=0;m<3;m++)
                  {
                         sprintf (&cipherString[m*2],"%02x", (unsigned
int)hashString[l]);                    if (m ==2)
                              {
                                   cipherString[5]=0;

                                   l=loop_saver;
                                   break;
                              }
                         l=l+1;
                    }
                    y = (int) strlen(cipherString);
                    decimal = 0;
                    k=0;
                    int ch;
                    while(y-1 != 0)
                    {
                         ch = cipherString[k];
                         if('0' <= ch && ch <= '9')
                         {
                              decimal = decimal * 16;
                              decimal = decimal + (ch - '0');
                         }
                         else if('A' <= ch && ch <= 'F')
                         {
                            decimal = decimal * 16;
                            decimal = decimal + (ch - 'A')+10;
                         }
                         else if('a' <= ch && ch <= 'f')
                         {
                              decimal = decimal * 16;
                              decimal = decimal + (ch - 'a')+10;
                         }
                         else
                         {
                              decimal=0;
                              break;
                         }
                         y--;
                         k++;
                    }
                    file_counter+=1;
                     int found = 0;
                    char chartest[300] = {0x00};
                    if(decimal<125000)
                    {
                         chartest[0] = buffer[decimal];
                         if (isalnum(*chartest) || *chartest == '_')
                         {
                           content[index++] = buffer[decimal];
                         }
                    }
                    if(found == 0)
                      printf("");
               }

          }
          int h;
          begin_hmac=clock();
          char key2[] = "012345678";
          unsigned char* digest;
          char mdString[40];
          sprintf(content,"%s%d",content,snsr1_reading_i);
          digest = HMAC(EVP_sha1(), key2, strlen(key2), (unsigned
char*)content, strlen(content), NULL, NULL);

          for(h = 0; h < 20; h++)
          {
               sprintf (&mdString[h*2], "%02x", (unsigned
int)digest[h]);
          }
        int ret=strncmp(mdString,msg_digest,40);
        if(ret==0)
          {
               printf("\n\nDigest match.\n");
          }
          else
          {
            printf("\n\nDigest does not match. Packet dropped.\nPress
any key to continue..\n");
        }
        printf("\nmdString: %s",mdString);

          end_hmac=clock();
          time_spent_hmac+= (double)(end_hmac - begin_hmac);
          // hmac ends here
          end = clock();
          time_spent = (double)(end - begin);
          double exectime = time_spent/CLOCKS_PER_SEC;
          long t1 = GetTimeMs64();
          char * f_result=0;

           printf("\nThe Execution Time in nano seconds is %d ",t1-t0);

  }
    getche();
}
```

DEV C++ CODE OF OWCAP (SKIPJACK) FOR SENSOR NODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
typedef unsigned char   byte;
typedef unsigned int    word32;

static const byte fTable[256] = {
     0xa3,0xd7,0x09,0x83,0xf8,0x48,0xf6,0xf4,0xb3,0x21,0x15,0x78,0x99,0
xb1,0xaf,0xf9,
     0xe7,0x2d,0x4d,0x8a,0xce,0x4c,0xca,0x2e,0x52,0x95,0xd9,0x1e,0x4e,0
x38,0x44,0x28,
     0x0a,0xdf,0x02,0xa0,0x17,0xf1,0x60,0x68,0x12,0xb7,0x7a,0xc3,0xe9,0
xfa,0x3d,0x53,
     0x96,0x84,0x6b,0xba,0xf2,0x63,0x9a,0x19,0x7c,0xae,0xe5,0xf5,0xf7,0
x16,0x6a,0xa2,
     0x39,0xb6,0x7b,0x0f,0xc1,0x93,0x81,0x1b,0xee,0xb4,0x1a,0xea,0xd0,0
x91,0x2f,0xb8,
     0x55,0xb9,0xda,0x85,0x3f,0x41,0xbf,0xe0,0x5a,0x58,0x80,0x5f,0x66,0
x0b,0xd8,0x90,
     0x35,0xd5,0xc0,0xa7,0x33,0x06,0x65,0x69,0x45,0x00,0x94,0x56,0x6d,0
x98,0x9b,0x76,
     0x97,0xfc,0xb2,0xc2,0xb0,0xfe,0xdb,0x20,0xe1,0xeb,0xd6,0xe4,0xdd,0
x47,0x4a,0x1d,
     0x42,0xed,0x9e,0x6e,0x49,0x3c,0xcd,0x43,0x27,0xd2,0x07,0xd4,0xde,0
xc7,0x67,0x18,
     0x89,0xcb,0x30,0x1f,0x8d,0xc6,0x8f,0xaa,0xc8,0x74,0xdc,0xc9,0x5d,0
x5c,0x31,0xa4,
     0x70,0x88,0x61,0x2c,0x9f,0x0d,0x2b,0x87,0x50,0x82,0x54,0x64,0x26,0
x7d,0x03,0x40,
     0x34,0x4b,0x1c,0x73,0xd1,0xc4,0xfd,0x3b,0xcc,0xfb,0x7f,0xab,0xe6,0
x3e,0x5b,0xa5,
     0xad,0x04,0x23,0x9c,0x14,0x51,0x22,0xf0,0x29,0x79,0x71,0x7e,0xff,0
x8c,0x0e,0xe2,
     0x0c,0xef,0xbc,0x72,0x75,0x6f,0x37,0xa1,0xec,0xd3,0x8e,0x62,0x8b,0
x86,0x10,0xe8,
     0x08,0x77,0x11,0xbe,0x92,0x4f,0x24,0xc5,0x32,0x36,0x9d,0xcf,0xf3,0
xa6,0xbb,0xac,
     0x5e,0x6c,0xa9,0x13,0x57,0x25,0xb5,0xe3,0xbd,0xa8,0x3a,0x01,0x05,0
x59,0x2a,0x46
};

#define g(tab, w, i, j, k, l) \
{ \
     w ^= (word32)tab[i][w & 0xff] << 8; \
     w ^= (word32)tab[j][w >>   8]; \
     w ^= (word32)tab[k][w & 0xff] << 8; \
     w ^= (word32)tab[l][w >>   8]; \
}

#define g0(tab, w) g(tab, w, 0, 1, 2, 3)
#define g1(tab, w) g(tab, w, 4, 5, 6, 7)
```

```
#define g2(tab, w) g(tab, w, 8, 9, 0, 1)
#define g3(tab, w) g(tab, w, 2, 3, 4, 5)
#define g4(tab, w) g(tab, w, 6, 7, 8, 9)

/**
 * The inverse of the G permutation.
 */
#define h(tab, w, i, j, k, l) \
{ \
     w ^= (word32)tab[l][w >>   8]; \
     w ^= (word32)tab[k][w & 0xff] << 8; \
     w ^= (word32)tab[j][w >>   8]; \
     w ^= (word32)tab[i][w & 0xff] << 8; \
}

#define h0(tab, w) h(tab, w, 0, 1, 2, 3)
#define h1(tab, w) h(tab, w, 4, 5, 6, 7)
#define h2(tab, w) h(tab, w, 8, 9, 0, 1)
#define h3(tab, w) h(tab, w, 2, 3, 4, 5)
#define h4(tab, w) h(tab, w, 6, 7, 8, 9)

void makeKey(byte key[10], byte tab[10][256]) {
          int i;
     for (i = 0; i < 10; i++) {
          byte *t = tab[i], k = key[i];
          int c;
          for (c = 0; c < 256; c++) {
               t[c] = fTable[c ^ k];
          }
     }
}

void encrypt(byte tab[10][256], byte in[8], byte out[8]) {
     word32 w1, w2, w3, w4;
//getch();
     w1 = (in[0] << 8) + in[1];
     w2 = (in[2] << 8) + in[3];
     w3 = (in[4] << 8) + in[5];
     w4 = (in[6] << 8) + in[7];

     /* stepping rule A: */
     g0(tab, w1); w4 ^= w1 ^ 1;
     g1(tab, w4); w3 ^= w4 ^ 2;
     g2(tab, w3); w2 ^= w3 ^ 3;
     g3(tab, w2); w1 ^= w2 ^ 4;
     g4(tab, w1); w4 ^= w1 ^ 5;
     g0(tab, w4); w3 ^= w4 ^ 6;
     g1(tab, w3); w2 ^= w3 ^ 7;
     g2(tab, w2); w1 ^= w2 ^ 8;

     /* stepping rule B: */
     w2 ^= w1 ^  9; g3(tab, w1);
     w1 ^= w4 ^ 10; g4(tab, w4);
     w4 ^= w3 ^ 11; g0(tab, w3);
     w3 ^= w2 ^ 12; g1(tab, w2);
     w2 ^= w1 ^ 13; g2(tab, w1);
     w1 ^= w4 ^ 14; g3(tab, w4);
     w4 ^= w3 ^ 15; g4(tab, w3);
     w3 ^= w2 ^ 16; g0(tab, w2);

     /* stepping rule A: */
     g1(tab, w1); w4 ^= w1 ^ 17;
     g2(tab, w4); w3 ^= w4 ^ 18;
     g3(tab, w3); w2 ^= w3 ^ 19;
     g4(tab, w2); w1 ^= w2 ^ 20;
     g0(tab, w1); w4 ^= w1 ^ 21;
     g1(tab, w4); w3 ^= w4 ^ 22;
     g2(tab, w3); w2 ^= w3 ^ 23;
     g3(tab, w2); w1 ^= w2 ^ 24;

     /* stepping rule B: */
     w2 ^= w1 ^ 25; g4(tab, w1);
     w1 ^= w4 ^ 26; g0(tab, w4);
     w4 ^= w3 ^ 27; g1(tab, w3);
     w3 ^= w2 ^ 28; g2(tab, w2);
     w2 ^= w1 ^ 29; g3(tab, w1);
     w1 ^= w4 ^ 30; g4(tab, w4);
     w4 ^= w3 ^ 31; g0(tab, w3);
     w3 ^= w2 ^ 32; g1(tab, w2);

     out[0] = (byte)(w1 >> 8); out[1] = (byte)w1;
     out[2] = (byte)(w2 >> 8); out[3] = (byte)w2;
     out[4] = (byte)(w3 >> 8); out[5] = (byte)w3;
     out[6] = (byte)(w4 >> 8); out[7] = (byte)w4;

}

void decrypt(byte tab[10][256], byte in[8], byte out[8]) {
     word32 w1, w2, w3, w4;

     w1 = (in[0] << 8) + in[1];
     w2 = (in[2] << 8) + in[3];
     w3 = (in[4] << 8) + in[5];
     w4 = (in[6] << 8) + in[7];

     /* stepping rule A: */
     h1(tab, w2); w3 ^= w2 ^ 32;
     h0(tab, w3); w4 ^= w3 ^ 31;
     h4(tab, w4); w1 ^= w4 ^ 30;
     h3(tab, w1); w2 ^= w1 ^ 29;
     h2(tab, w2); w3 ^= w2 ^ 28;
     h1(tab, w3); w4 ^= w3 ^ 27;
     h0(tab, w4); w1 ^= w4 ^ 26;
     h4(tab, w1); w2 ^= w1 ^ 25;

     /* stepping rule B: */
     w1 ^= w2 ^ 24; h3(tab, w2);
     w2 ^= w3 ^ 23; h2(tab, w3);
     w3 ^= w4 ^ 22; h1(tab, w4);
     w4 ^= w1 ^ 21; h0(tab, w1);
     w1 ^= w2 ^ 20; h4(tab, w2);
     w2 ^= w3 ^ 19; h3(tab, w3);
     w3 ^= w4 ^ 18; h2(tab, w4);
     w4 ^= w1 ^ 17; h1(tab, w1);

     /* stepping rule A: */
     h0(tab, w2); w3 ^= w2 ^ 16;
     h4(tab, w3); w4 ^= w3 ^ 15;
     h3(tab, w4); w1 ^= w4 ^ 14;
     h2(tab, w1); w2 ^= w1 ^ 13;
     h1(tab, w2); w3 ^= w2 ^ 12;
     h0(tab, w3); w4 ^= w3 ^ 11;
     h4(tab, w4); w1 ^= w4 ^ 10;
     h3(tab, w1); w2 ^= w1 ^  9;

     /* stepping rule B: */
     w1 ^= w2 ^ 8; h2(tab, w2);
     w2 ^= w3 ^ 7; h1(tab, w3);
     w3 ^= w4 ^ 6; h0(tab, w4);
     w4 ^= w1 ^ 5; h4(tab, w1);
     w1 ^= w2 ^ 4; h3(tab, w2);
     w2 ^= w3 ^ 3; h2(tab, w3);
     w3 ^= w4 ^ 2; h1(tab, w4);
     w4 ^= w1 ^ 1; h0(tab, w1);

     out[0] = (byte)(w1 >> 8); out[1] = (byte)w1;
     out[2] = (byte)(w2 >> 8); out[3] = (byte)w2;
     out[4] = (byte)(w3 >> 8); out[5] = (byte)w3;
     out[6] = (byte)(w4 >> 8); out[7] = (byte)w4;

}

long GetTimeMs64()
{
 FILETIME ft;
 LARGE_INTEGER li;

 /* Get the amount of 100 nano seconds intervals elapsed since January
1, 1601 (UTC) and copy it
  * to a LARGE_INTEGER structure. */
 GetSystemTimeAsFileTime(&ft);
 li.LowPart = ft.dwLowDateTime;
 li.HighPart = ft.dwHighDateTime;

 long ret = li.QuadPart;
 ret -= 116444736000000000LL;
 return ret;

}
main( )
{
    long t0 = GetTimeMs64();
   time_t now;
    struct tm * ptm;
    now = time(NULL);
    ptm = gmtime ( &now );
    char t_stamp[10];
    char mon1[2];
    char day[2];
    char hour[2];
    char minutes[2];
    char seconds[2];
    if((ptm->tm_mon+1)<10)
    {
        sprintf(t_stamp,"0%d",ptm->tm_mon+1);
    }
    else
    {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

    if(ptm->tm_mday<10)
       { sprintf(day,"0%d",ptm->tm_mday);}
    else
        {sprintf(day,"%d",ptm->tm_mday);}
        strcat(t_stamp,day);
  if((ptm->tm_hour+5)<10)
      {sprintf(hour,"0%d",ptm->tm_hour+5);}
  else
      {sprintf(hour,"%d",ptm->tm_hour+5);}
      strcat(t_stamp,hour);
   if(ptm->tm_min<10)
      {sprintf(minutes,"0%d",ptm->tm_min);}
  else
      {sprintf(minutes,"%d",ptm->tm_min);}
      strcat(t_stamp,minutes);
   if(ptm->tm_sec<10)
      {sprintf(seconds,"0%d",ptm->tm_sec);}
  else
      {sprintf(seconds,"%d",ptm->tm_sec);}
      strcat(t_stamp,seconds);
    printf("\nt_stamp: %s\n",t_stamp);
  unsigned long time_stamp=atoi(t_stamp);
  unsigned int sensor_reading=ptm->tm_sec%10+1;
  char s_reading[2];
  sprintf(s_reading,"%d",sensor_reading);

  printf("time: %d\n",time_stamp);
clock_t begin, end;
clock_t begin_sha,end_sha;
clock_t begin_rc5,end_rc5;
clock_t begin_file,end_file;
clock_t begin_hmac,end_hmac;
double time_spent;
double time_spent_sha=0;
double time_spent_rc5=0;
double time_spent_file=0;
double time_spent_hmac=0;
  begin = clock();
  int j,i,y,k,l,m,decimal;
  int index = 0;
  char content[300] = {0x00};

  FILE *fp;
   char * buffer = 0;
   long length;
   FILE * f = fopen ("memread.txt", "rb");

   if (f)
   {
     fseek (f, 0, SEEK_END);
     length = ftell (f);
     fseek (f, 0, SEEK_SET);
     buffer = malloc (length);
     if (buffer)
     {
        fread (buffer, 1, length, f);
     }
     fclose (f);
   }
  unsigned char ibuf[32];
  unsigned char obuf[64];
  char hashString[64];
  char cipherString[64];
  int hexadecimal;
   int file_counter=0;
```

```
    int srno= 65535;
    int loop_saver;
    for(i = 0; i<20; i++)
    {
        begin_sha=clock();
        long combo = i + srno+time_stamp;
        sprintf(ibuf, "%d", combo);
        SHA256(ibuf,strlen(ibuf),obuf);
        for (j=0;j<32;j++)
        { sprintf (&hashString[j*2],"%02x", (unsigned int)obuf[j]);

        }

        end_sha=clock();
        time_spent_sha+= (double)(end_sha - begin_sha);
        int i;
     byte inp[16]       = { 0x33, 0x22, 0x11, 0x00, 0xdd, 0xcc, 0xbb,
0xaa,0x33, 0x22, 0x11, 0x00, 0xdd, 0xcc, 0xbb, 0xaa };
     byte key[10]   = { 0x00, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33,
0x22, 0x11 };
     byte enc[16], dec[8];
     byte chk[8]   = { 0x25, 0x87, 0xca, 0xe2, 0x7a, 0x12, 0xd3, 0x00
};
     byte tab[10][256];
          makeKey(key, tab);

     encrypt(tab, hashString, hashString);
          end_rc5=clock();
          time_spent_rc5+= (double)(end_rc5 - begin_rc5);
          for (l=0;l<32;l++)
          {
           loop_saver=l;
             for (m=0;m<3;m++)
             {
                    sprintf (&cipherString[m*2],"%02x", (unsigned
int)hashString[l]);
                    if (m ==2)
                         {
                              cipherString[5]=0;

                              l=loop_saver;
                              break;
                         }
                    l=l+1;
               }
               y = (int) strlen(cipherString);
               decimal = 0;
               k=0;
               int ch;
               while(y-1 != 0)
               {
                    ch = cipherString[k];
                    if('0' <= ch && ch <= '9')
                    {
                         decimal = decimal * 16;
                         decimal = decimal + (ch - '0');
                    }
                    else if('A' <= ch && ch <= 'F')
                    {
                       decimal = decimal * 16;
                       decimal = decimal + (ch - 'A')+10;
                    }
                    else if('a' <= ch && ch <= 'f')
                    {
                         decimal = decimal * 16;
                         decimal = decimal + (ch - 'a')+10;
                    }
                    else
                    {
                         decimal=0;
                         break;
                    }
                    y--;
                    k++;
               }

               int found = 0;
               char chartest[300] = {0x00};
               if(decimal<125000)
               {
                    chartest[0] = buffer[decimal];
                    if (isalnum(*chartest) || *chartest == '_')
                    {
                      content[index++] = buffer[decimal];
          }
               }
               if(found == 0)
                 printf("");
          }

    }
    int h;
     begin_hmac=clock();
    char key2[] = "012345678";
    unsigned char* digest;
    char mdString[40];
    strcat(content,s_reading);
    digest = HMAC(EVP_sha1(), key2, strlen(key2), (unsigned
char*)content, strlen(content), NULL, NULL);

    for(h = 0; h < 20; h++)
    {
          sprintf (&mdString[h*2], "%02x", (unsigned int)digest[h]);
    }
    printf("\nmdString: %s,t_stamp: %s,sensor_reading:
%s",mdString,t_stamp,s_reading);

     end_hmac=clock();
     time_spent_hmac+= (double)(end_hmac - begin_hmac);
    // hmac ends here
     end = clock();
     time_spent = (double)(end - begin);
    double exectime = time_spent/CLOCKS_PER_SEC;
    long t1 = GetTimeMs64();
    char * f_result=0;
    printf("\nThe Execution Time in nano seconds is %d ",t1-t0);
    FILE *fh = fopen("out.dat", "w");
    fprintf(fh,"%s,%s,%s\n",mdString,t_stamp,s_reading);
    close(fh);
    getch();
}
```

DEV C++ CODE OF OWCAP (SKIPJACK) FOR VERIFIER

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
typedef unsigned char   byte;
typedef unsigned int    word32;

static const byte fTable[256] = {
     0xa3,0xd7,0x09,0x83,0xf8,0x48,0xf6,0xf4,0xb3,0x21,0x15,0x78,0x99,0
xb1,0xaf,0xf9,
     0xe7,0x2d,0x4d,0x8a,0xce,0x4c,0xca,0x2e,0x52,0x95,0xd9,0x1e,0x4e,0
x38,0x44,0x28,
     0x0a,0xdf,0x02,0xa0,0x17,0xf1,0x60,0x68,0x12,0xb7,0x7a,0xc3,0xe9,0
xfa,0x3d,0x53,
     0x96,0x84,0x6b,0xba,0xf2,0x63,0x9a,0x19,0x7c,0xae,0xe5,0xf5,0xf7,0
x16,0x6a,0xa2,
     0x39,0xb6,0x7b,0x0f,0xc1,0x93,0x81,0x1b,0xee,0xb4,0x1a,0xea,0xd0,0
x91,0x2f,0xb8,
     0x55,0xb9,0xda,0x85,0x3f,0x41,0xbf,0xe0,0x5a,0x58,0x80,0x5f,0x66,0
x0b,0xd8,0x90,
     0x35,0xd5,0xc0,0xa7,0x33,0x06,0x65,0x69,0x45,0x00,0x94,0x56,0x6d,0
x98,0x9b,0x76,
     0x97,0xfc,0xb2,0xc2,0xb0,0xfe,0xdb,0x20,0xe1,0xeb,0xd6,0xe4,0xdd,0
x47,0x4a,0x1d,
     0x42,0xed,0x9e,0x6e,0x49,0x3c,0xcd,0x43,0x27,0xd2,0x07,0xd4,0xde,0
xc7,0x67,0x18,
     0x89,0xcb,0x30,0x1f,0x8d,0xc6,0x8f,0xaa,0xc8,0x74,0xdc,0xc9,0x5d,0
x5c,0x31,0xa4,
     0x70,0x88,0x61,0x2c,0x9f,0x0d,0x2b,0x87,0x50,0x82,0x54,0x64,0x26,0
x7d,0x03,0x40,
     0x34,0x4b,0x1c,0x73,0xd1,0xc4,0xfd,0x3b,0xcc,0xfb,0x7f,0xab,0xe6,0
x3e,0x5b,0xa5,
     0xad,0x04,0x23,0x9c,0x14,0x51,0x22,0xf0,0x29,0x79,0x71,0x7e,0xff,0
x8c,0x0e,0xe2,
     0x0c,0xef,0xbc,0x72,0x75,0x6f,0x37,0xa1,0xec,0xd3,0x8e,0x62,0x8b,0
x86,0x10,0xe8,
     0x08,0x77,0x11,0xbe,0x92,0x4f,0x24,0xc5,0x32,0x36,0x9d,0xcf,0xf3,0
xa6,0xbb,0xac,
     0x5e,0x6c,0xa9,0x13,0x57,0x25,0xb5,0xe3,0xbd,0xa8,0x3a,0x01,0x05,0
x59,0x2a,0x46
};

/**
 * The key-dependent permutation G on V^16 is a four-round Feistel
network.
 * The round function is a fixed byte-substitution table (permutation
on V^8),
 * the F-table. */
#define g(tab, w, i, j, k, l) \
{ \
     w ^= (word32)tab[i][w & 0xff] << 8; \
     w ^= (word32)tab[j][w >>   8]; \
     w ^= (word32)tab[k][w & 0xff] << 8; \
     w ^= (word32)tab[l][w >>   8]; \
}

#define g0(tab, w) g(tab, w, 0, 1, 2, 3)
#define g1(tab, w) g(tab, w, 4, 5, 6, 7)
#define g2(tab, w) g(tab, w, 8, 9, 0, 1)
#define g3(tab, w) g(tab, w, 2, 3, 4, 5)
#define g4(tab, w) g(tab, w, 6, 7, 8, 9)

/**
 * The inverse of the G permutation.
 */
#define h(tab, w, i, j, k, l) \
{ \
     w ^= (word32)tab[l][w >>   8]; \
     w ^= (word32)tab[k][w & 0xff] << 8; \
     w ^= (word32)tab[j][w >>   8]; \
     w ^= (word32)tab[i][w & 0xff] << 8; \
}

#define h0(tab, w) h(tab, w, 0, 1, 2, 3)
#define h1(tab, w) h(tab, w, 4, 5, 6, 7)
#define h2(tab, w) h(tab, w, 8, 9, 0, 1)
#define h3(tab, w) h(tab, w, 2, 3, 4, 5)
#define h4(tab, w) h(tab, w, 6, 7, 8, 9)

void makeKey(byte key[10], byte tab[10][256]) {
     /* tab[i][c] = fTable[c ^ key[i]] */
     int i;
     for (i = 0; i < 10; i++) {
          byte *t = tab[i], k = key[i];
          int c;
          for (c = 0; c < 256; c++) {
               t[c] = fTable[c ^ k];
          }
     }
}

void encrypt(byte tab[10][256], byte in[8], byte out[8]) {
     word32 w1, w2, w3, w4;
//getch();
     w1 = (in[0] << 8) + in[1];
     w2 = (in[2] << 8) + in[3];
     w3 = (in[4] << 8) + in[5];
     w4 = (in[6] << 8) + in[7];

     /* stepping rule A: */
     g0(tab, w1); w4 ^= w1 ^ 1;
     g1(tab, w4); w3 ^= w4 ^ 2;
     g2(tab, w3); w2 ^= w3 ^ 3;
     g3(tab, w2); w1 ^= w2 ^ 4;
     g4(tab, w1); w4 ^= w1 ^ 5;
     g0(tab, w4); w3 ^= w4 ^ 6;
     g1(tab, w3); w2 ^= w3 ^ 7;
     g2(tab, w2); w1 ^= w2 ^ 8;

     /* stepping rule B: */
     w2 ^= w1 ^  9; g3(tab, w1);
     w1 ^= w4 ^ 10; g4(tab, w4);
     w4 ^= w3 ^ 11; g0(tab, w3);
     w3 ^= w2 ^ 12; g1(tab, w2);
     w2 ^= w1 ^ 13; g2(tab, w1);
     w1 ^= w4 ^ 14; g3(tab, w4);
     w4 ^= w3 ^ 15; g4(tab, w3);
     w3 ^= w2 ^ 16; g0(tab, w2);

     /* stepping rule A: */
     g1(tab, w1); w4 ^= w1 ^ 17;
     g2(tab, w4); w3 ^= w4 ^ 18;
     g3(tab, w3); w2 ^= w3 ^ 19;
     g4(tab, w2); w1 ^= w2 ^ 20;
     g0(tab, w1); w4 ^= w1 ^ 21;
     g1(tab, w4); w3 ^= w4 ^ 22;
     g2(tab, w3); w2 ^= w3 ^ 23;
     g3(tab, w2); w1 ^= w2 ^ 24;

     /* stepping rule B: */
     w2 ^= w1 ^ 25; g4(tab, w1);
     w1 ^= w4 ^ 26; g0(tab, w4);
     w4 ^= w3 ^ 27; g1(tab, w3);
     w3 ^= w2 ^ 28; g2(tab, w2);
     w2 ^= w1 ^ 29; g3(tab, w1);
     w1 ^= w4 ^ 30; g4(tab, w4);
     w4 ^= w3 ^ 31; g0(tab, w3);
     w3 ^= w2 ^ 32; g1(tab, w2);

     out[0] = (byte)(w1 >> 8); out[1] = (byte)w1;
     out[2] = (byte)(w2 >> 8); out[3] = (byte)w2;
     out[4] = (byte)(w3 >> 8); out[5] = (byte)w3;
     out[6] = (byte)(w4 >> 8); out[7] = (byte)w4;

}

void decrypt(byte tab[10][256], byte in[8], byte out[8]) {
     word32 w1, w2, w3, w4;

     w1 = (in[0] << 8) + in[1];
     w2 = (in[2] << 8) + in[3];
     w3 = (in[4] << 8) + in[5];
     w4 = (in[6] << 8) + in[7];

     /* stepping rule A: */
     h1(tab, w2); w3 ^= w2 ^ 32;
     h0(tab, w3); w4 ^= w3 ^ 31;
     h4(tab, w4); w1 ^= w4 ^ 30;
     h3(tab, w1); w2 ^= w1 ^ 29;
     h2(tab, w2); w3 ^= w2 ^ 28;
     h1(tab, w3); w4 ^= w3 ^ 27;
     h0(tab, w4); w1 ^= w4 ^ 26;
     h4(tab, w1); w2 ^= w1 ^ 25;

     /* stepping rule B: */
     w1 ^= w2 ^ 24; h3(tab, w2);
     w2 ^= w3 ^ 23; h2(tab, w3);
     w3 ^= w4 ^ 22; h1(tab, w4);
     w4 ^= w1 ^ 21; h0(tab, w1);
     w1 ^= w2 ^ 20; h4(tab, w2);
     w2 ^= w3 ^ 19; h3(tab, w3);
     w3 ^= w4 ^ 18; h2(tab, w4);
     w4 ^= w1 ^ 17; h1(tab, w1);
     /* stepping rule A: */
     h0(tab, w2); w3 ^= w2 ^ 16;
     h4(tab, w3); w4 ^= w3 ^ 15;
     h3(tab, w4); w1 ^= w4 ^ 14;
     h2(tab, w1); w2 ^= w1 ^ 13;
     h1(tab, w2); w3 ^= w2 ^ 12;
     h0(tab, w3); w4 ^= w3 ^ 11;
     h4(tab, w4); w1 ^= w4 ^ 10;
     h3(tab, w1); w2 ^= w1 ^  9;

     /* stepping rule B: */
     w1 ^= w2 ^ 8; h2(tab, w2);
     w2 ^= w3 ^ 7; h1(tab, w3);
     w3 ^= w4 ^ 6; h0(tab, w4);
     w4 ^= w1 ^ 5; h4(tab, w1);
     w1 ^= w2 ^ 4; h3(tab, w2);
     w2 ^= w3 ^ 3; h2(tab, w3);
     w3 ^= w4 ^ 2; h1(tab, w4);
     w4 ^= w1 ^ 1; h0(tab, w1);

     out[0] = (byte)(w1 >> 8); out[1] = (byte)w1;
     out[2] = (byte)(w2 >> 8); out[3] = (byte)w2;
     out[4] = (byte)(w3 >> 8); out[5] = (byte)w3;
     out[6] = (byte)(w4 >> 8); out[7] = (byte)w4;

}

long GetTimeMs64()
{
 FILETIME ft;
 LARGE_INTEGER li;

 /* Get the amount of 100 nano seconds intervals elapsed since January
1, 1601 (UTC) and copy it
  * to a LARGE_INTEGER structure. */
 GetSystemTimeAsFileTime(&ft);
 li.LowPart = ft.dwLowDateTime;
 li.HighPart = ft.dwHighDateTime;

 long ret = li.QuadPart;
 ret -= 116444736000000000LL;
 return ret;

}
main( )
{
     char * rx_data = 0;
     long length;
     FILE * fp = fopen ("out.dat", "rb");

     if (fp)
     {
       fseek (fp, 0, SEEK_END);
       length = ftell (fp);
       fseek (fp, 0, SEEK_SET);
       rx_data = malloc (length);
       if (rx_data)
       {
          fread (rx_data, 1, length, fp);
       }
       fclose (fp);
     }
     char * msg_digest;
     char * time_n1;
     char * snsr1_reading;
int snsr1_reading_i;
     msg_digest = strtok(rx_data, ",");
     time_n1 = strtok(NULL, ",");
      snsr1_reading = strtok(NULL, ",");

     snsr1_reading = strtok(snsr1_reading, "\n");

    printf("\nReceived data\nMsg Digest n1: %s", msg_digest);
    printf("\nTime n1:%s", time_n1);
     printf("\nSensor reading n1: %s\n", snsr1_reading);
     snsr1_reading_i=atoi(snsr1_reading);
    long t0 = GetTimeMs64();
    time_t now;
    struct tm * ptm;
    now = time(NULL);
    ptm = gmtime ( &now );
    char t_stamp[10];

     char mon1[2];
    char day[2];
    char hour[2];
    char minutes[2];
    char seconds[2];
    if((ptm->tm_mon+1)<10)
    {
        sprintf(t_stamp,"0%d",ptm->tm_mon+1);
    }
    else
    {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

    if(ptm->tm_mday<10)
       { sprintf(day,"0%d",ptm->tm_mday);}
    else
        {sprintf(day,"%d",ptm->tm_mday);}
          strcat(t_stamp,day);
    if((ptm->tm_hour+5)<10)
        {sprintf(hour,"0%d",ptm->tm_hour+5);}
    else
        {sprintf(hour,"%d",ptm->tm_hour+5);}
        strcat(t_stamp,hour);
     if(ptm->tm_min<10)
        {sprintf(minutes,"0%d",ptm->tm_min);}
    else
        {sprintf(minutes,"%d",ptm->tm_min);}
        strcat(t_stamp,minutes);
     if(ptm->tm_sec<10)
        {sprintf(seconds,"0%d",ptm->tm_sec);}
    else
        {sprintf(seconds,"%d",ptm->tm_sec);}
        strcat(t_stamp,seconds);
    unsigned long time_stamp=atoi(t_stamp);
    unsigned long time_stamp_n1=atoi(time_n1);
    printf("current time:%d",time_stamp);
    int time_difference=time_stamp-time_stamp_n1;
    printf("\ntime different: %d",time_difference);
    if(time_difference<10)
    {
         printf("\nTime difference is out of acceptable range. Packet
dropped!\nPress any key to continue..");
          getch();
          return 1;
    }
    else if(snsr1_reading_i<=0 || snsr1_reading_i>10)
    {
         printf("\nSensor range is out of acceptable range. Packet
dropped!\nPress any key to continue..");
          getch();
          return 1;
    }
    else
    {
         clock_t begin, end;
         clock_t begin_sha,end_sha;
         clock_t begin_rc5,end_rc5;
         clock_t begin_file,end_file;
         clock_t begin_hmac,end_hmac;
         double time_spent;
         double time_spent_sha=0;
         double time_spent_rc5=0;
         double time_spent_file=0;
         double time_spent_hmac=0;
         begin = clock();
         int j,i,y,k,l,m,decimal;
         int index = 0;
         char content[300] = {0x00};

         char * buffer = 0;
         FILE * f = fopen ("memread.txt", "rb");

         if (f)
         {
           fseek (f, 0, SEEK_END);
           length = ftell (f);
           fseek (f, 0, SEEK_SET);
           buffer = malloc (length);
           if (buffer)
           {
              fread (buffer, 1, length, f);
           }
           fclose (f);
         }
         unsigned char ibuf[32];
         unsigned char obuf[64];
         char hashString[64];
         char cipherString[64];
         int hexadecimal;
         int file_counter=0;
         int srno= 65535;
         int loop_saver;
         for(i = 0; i<20; i++)
         {
              begin_sha=clock();
              long combo = i + srno+time_stamp_n1;
              sprintf(ibuf, "%d", combo);
              SHA256(ibuf,strlen(ibuf),obuf);
              for (j=0;j<32;j++)
              { sprintf (&hashString[j*2],"%02x", (unsigned
int)obuf[j]);
```

```
            }

            end_sha=clock();
            time_spent_sha+= (double)(end_sha - begin_sha);
            int i;
            byte inp[16]       = { 0x33, 0x22, 0x11, 0x00, 0xdd,
0xcc, 0xbb, 0xaa,0x33, 0x22, 0x11, 0x00, 0xdd, 0xcc, 0xbb, 0xaa };
            byte key[10]   = { 0x00, 0x99, 0x88, 0x77, 0x66, 0x55,
0x44, 0x33, 0x22, 0x11 };
            byte enc[16], dec[8];
            byte chk[8]    = { 0x25, 0x87, 0xca, 0xe2, 0x7a, 0x12,
0xd3, 0x00 };
            byte tab[10][256];
            makeKey(key, tab);

            encrypt(tab, hashString, hashString);
            end_rc5=clock();
            time_spent_rc5+= (double)(end_rc5 - begin_rc5);
            for (l=0;l<32;l++)
            {
               loop_saver=l;
               for (m=0;m<3;m++)
               {
                     sprintf (&cipherString[m*2],"%02x", (unsigned
int)hashString[l]);
                     if (m ==2)
                         {
                             cipherString[5]=0;

                             l=loop_saver;
                             break;
                         }
                     l=l+1;
                }
                y = (int) strlen(cipherString);
                decimal = 0;
                k=0;
                int ch;
                while(y-1 != 0)
                {
                     ch = cipherString[k];
                     if('0' <= ch && ch <= '9')
                     {
                         decimal = decimal * 16;
                         decimal = decimal + (ch - '0');
                     }
                     else if('A' <= ch && ch <= 'F')
                     {
                        decimal = decimal * 16;
                        decimal = decimal + (ch - 'A')+10;
                     }
                     else if('a' <= ch && ch <= 'f')
                     {
                         decimal = decimal * 16;
                         decimal = decimal + (ch - 'a')+10;
                     }
                     else
                     {
                         decimal=0;
                         break;
                         }
                         y--;
                         k++;
                    }
                    int found = 0;
                    char chartest[300] = {0x00};
                    if(decimal<125000)
                    {
                         chartest[0] = buffer[decimal];
                         if (isalnum(*chartest) || *chartest == '_')
                         {
                           content[index++] = buffer[decimal];
                         }
                    }
                    if(found == 0)
                      printf("");
               }

          }
          int h;
          begin_hmac=clock();
          char key2[] = "012345678";
          unsigned char* digest;
          char mdString[40];
          sprintf(content,"%s%d",content,snsr1_reading_i);
          digest = HMAC(EVP_sha1(), key2, strlen(key2), (unsigned
char*)content, strlen(content), NULL, NULL);
          for(h = 0; h < 20; h++)
          {
               sprintf (&mdString[h*2], "%02x", (unsigned
int)digest[h]);
          }
                    int ret=strncmp(mdString,msg_digest,40);
        if(ret==0)
          {
               printf("\n\nDigest match.\n");
          }
          else
          {
            printf("\n\nDigest does not match. Packet dropped.\nPress
any key to continue..\n");
        }
          end_hmac=clock();
          time_spent_hmac+= (double)(end_hmac - begin_hmac);
          // hmac ends here
          end = clock();
          time_spent = (double)(end - begin);
          double exectime = time_spent/CLOCKS_PER_SEC;
          long t1 = GetTimeMs64();
          char * f_result=0;
           printf("\nThe Execution Time in nano seconds is %d ",t1-t0);

  }
    getche();
}
```

DEV C++ CODE OF OWCAP (SHA-2) FOR SENSOR NODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <openssl/sha.h>
#include <openssl/rc4.h>
#include <openssl/bio.h>
#include <openssl/hmac.h>
#include <conio.h>
#include <windows.h>
long GetTimeMs64()
{
 FILETIME ft;
 LARGE_INTEGER li;
GetSystemTimeAsFileTime(&ft);
 li.LowPart = ft.dwLowDateTime;
 li.HighPart = ft.dwHighDateTime;

 long ret = li.QuadPart;
 ret -= 116444736000000000LL;

 return ret;

}
main( )
{
    long t0 = GetTimeMs64();
   time_t now;
    struct tm * ptm;
    now = time(NULL);
    ptm = gmtime ( &now );
    char t_stamp[10];
    char mon1[2];
    char day[2];
    char hour[2];
    char minutes[2];
    char seconds[2];
    if((ptm->tm_mon+1)<10)
    {
        sprintf(t_stamp,"0%d",ptm->tm_mon+1);
    }
    else
    {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

    if(ptm->tm_mday<10)
       { sprintf(day,"0%d",ptm->tm_mday);}
    else
        {sprintf(day,"%d",ptm->tm_mday);}
          strcat(t_stamp,day);
    if((ptm->tm_hour+5)<10)
        {sprintf(hour,"0%d",ptm->tm_hour+5);}
    else
        {sprintf(hour,"%d",ptm->tm_hour+5);}
        strcat(t_stamp,hour);
     if(ptm->tm_min<10)
        {sprintf(minutes,"0%d",ptm->tm_min);}
    else
        {sprintf(minutes,"%d",ptm->tm_min);}
        strcat(t_stamp,minutes);
     if(ptm->tm_sec<10)
        {sprintf(seconds,"0%d",ptm->tm_sec);}
    else
        {sprintf(seconds,"%d",ptm->tm_sec);}
        strcat(t_stamp,seconds);
      printf("\nt_stamp: %s\n",t_stamp);
    unsigned long time_stamp=atoi(t_stamp);
    unsigned int sensor_reading=ptm->tm_sec%10+1;
    char s_reading[2];
    sprintf(s_reading,"%d",sensor_reading);

    printf("time: %d\n",time_stamp);
clock_t begin, end;
clock_t begin_sha,end_sha;
clock_t begin_rc5,end_rc5;
clock_t begin_file,end_file;
clock_t begin_hmac,end_hmac;
double time_spent;
double time_spent_sha=0;
double time_spent_rc5=0;
double time_spent_file=0;
double time_spent_hmac=0;
    begin = clock();
    int j,i,y,k,l,m,decimal;
    int index = 0;
    char content[400] = {0x00};

    FILE *fp;
     char * buffer = 0;
     long length;
     FILE * f = fopen ("memread.txt", "rb");

     if (f)
     {
       fseek (f, 0, SEEK_END);
       length = ftell (f);
       fseek (f, 0, SEEK_SET);
       buffer = malloc (length);
       if (buffer)
       {
          fread (buffer, 1, length, f);
       }
       fclose (f);
     }
    unsigned char ibuf[32];
    unsigned char obuf[64];
    char hashString[64];
    char cipherString[64];
    int hexadecimal;
     int file_counter=0;
    int srno= 65535;
    int loop_saver;
    for(i = 0; i<20; i++)
    {
        begin_sha=clock();
        long combo = i + srno+time_stamp;
        sprintf(ibuf, "%d", combo);
        SHA256(ibuf,strlen(ibuf),obuf);
        for (j=0;j<32;j++)
        { sprintf (&hashString[j*2],"%02x", (unsigned int)obuf[j]);

        }

        end_sha=clock();
        time_spent_sha+= (double)(end_sha - begin_sha);
        int i;
          for (l=0;l<32;l++)
          {
           loop_saver=l;
             for (m=0;m<3;m++)
             {
                   sprintf (&cipherString[m*2],"%02x", (unsigned
int)hashString[l]);
                   if (m ==2)
                        {
                             cipherString[5]=0;

                             l=loop_saver;
                             break;
                        }
                   l=l+1;
              }
              y = (int) strlen(cipherString);
              decimal = 0;
              k=0;
              int ch;
              while(y-1 != 0)
              {
                   ch = cipherString[k];
                   if('0' <= ch && ch <= '9')
                   {
                        decimal = decimal * 16;
                        decimal = decimal + (ch - '0');
                   }
                   else if('A' <= ch && ch <= 'F')
                   {
                      decimal = decimal * 16;
                      decimal = decimal + (ch - 'A')+10;
                   }
                   else if('a' <= ch && ch <= 'f')
                   {
                        decimal = decimal * 16;
                        decimal = decimal + (ch - 'a')+10;
                   }
                   else
                   {
                        decimal=0;
                        break;
                   }
                   y--;
                   k++;
              }

              int found = 0;
              char chartest[300] = {0x00};
              if(decimal<125000)
              {
                   chartest[0] = buffer[decimal];
                   if (isalnum(*chartest) || *chartest == '_')
                   {
                     content[index++] = buffer[decimal];
         }
              }

              if(found == 0)
                printf("");
         }

    }
    int h;
     begin_hmac=clock();
    char key2[] = "012345678";
    unsigned char* digest;
    char mdString[40];
    sprintf(content,"%s%d",content,sensor_reading);

    digest = HMAC(EVP_sha1(), key2, strlen(key2), (unsigned
char*)content, strlen(content), NULL, NULL);

    for(h = 0; h < 20; h++)
    {
         sprintf (&mdString[h*2], "%02x", (unsigned int)digest[h]);

    }
    printf("\nmdString: %s,t_stamp: %s,sensor_reading:
%s",mdString,t_stamp,s_reading);

     end_hmac=clock();
     time_spent_hmac+= (double)(end_hmac - begin_hmac);
    // hmac ends here
     end = clock();
     time_spent = (double)(end - begin);
    double exectime = time_spent/CLOCKS_PER_SEC;
    long t1 = GetTimeMs64();
    char * f_result=0;
     printf("\nThe Execution Time in nano seconds is %d ",t1-t0);
    FILE *fh = fopen("out.dat", "w");
    fprintf(fh,"%s,%s,%s\n",mdString,t_stamp,s_reading);
    close(fh);
    getch();
}
```

DEV C++ CODE OF OWCAP (SHA-2) FOR VERIFIER

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <openssl/sha.h>
#include <openssl/rc4.h>
#include <openssl/bio.h>
#include <openssl/hmac.h>
#include <conio.h>
#include <windows.h>
long GetTimeMs64()
{
 FILETIME ft;
 LARGE_INTEGER li;

 GetSystemTimeAsFileTime(&ft);
 li.LowPart = ft.dwLowDateTime;
 li.HighPart = ft.dwHighDateTime;

 long ret = li.QuadPart;
 ret -= 116444736000000000LL;

 return ret;

}
main( )
{
     char * rx_data = 0;
     long length;
     FILE * fp = fopen ("out.dat", "rb");

     if (fp)
     {
       fseek (fp, 0, SEEK_END);
       length = ftell (fp);
       fseek (fp, 0, SEEK_SET);
       rx_data = malloc (length);
       if (rx_data)
       {
          fread (rx_data, 1, length, fp);
       }
       fclose (fp);
     }
     char * msg_digest;
     char * time_n1;
     char * snsr1_reading;
int snsr1_reading_i;
     msg_digest = strtok(rx_data, ",");
     time_n1 = strtok(NULL, ",");
      snsr1_reading = strtok(NULL, ",");

     snsr1_reading = strtok(snsr1_reading, "\n");
```

```
    printf("\nReceived data\nMsg Digest n1: %s", msg_digest);
    printf("\nTime n1:%s", time_n1);
     printf("\nSensor reading n1: %s\n", snsr1_reading);
     snsr1_reading_i=atoi(snsr1_reading);
    long t0 = GetTimeMs64();
    time_t now;
    struct tm * ptm;
    now = time(NULL);
    ptm = gmtime ( &now );
    char t_stamp[10];

     char mon1[2];
    char day[2];
    char hour[2];
    char minutes[2];
    char seconds[2];
    if((ptm->tm_mon+1)<10)
    {
        sprintf(t_stamp,"0%d",ptm->tm_mon+1);
    }
    else
    {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

    if(ptm->tm_mday<10)
       { sprintf(day,"0%d",ptm->tm_mday);}
    else
        {sprintf(day,"%d",ptm->tm_mday);}
          strcat(t_stamp,day);
    if((ptm->tm_hour+5)<10)
        {sprintf(hour,"0%d",ptm->tm_hour+5);}
    else
        {sprintf(hour,"%d",ptm->tm_hour+5);}
        strcat(t_stamp,hour);
     if(ptm->tm_min<10)
        {sprintf(minutes,"0%d",ptm->tm_min);}
    else
        {sprintf(minutes,"%d",ptm->tm_min);}
        strcat(t_stamp,minutes);
     if(ptm->tm_sec<10)
        {sprintf(seconds,"0%d",ptm->tm_sec);}
    else
        {sprintf(seconds,"%d",ptm->tm_sec);}
        strcat(t_stamp,seconds);
    unsigned long time_stamp=atoi(t_stamp);
    unsigned long time_stamp_n1=atoi(time_n1);
    printf("current time:%d",time_stamp);
    int time_difference=time_stamp-time_stamp_n1;
    printf("\ntime difference: %d",time_difference);
    if(time_difference<10)
    {
         printf("\nTime difference is out of acceptable range. Packet
dropped!\nPress any key to continue..");
           getch();
           return 1;
    }
    else if(snsr1_reading_i<=0 || snsr1_reading_i>10)
    {
         printf("\nSensor range is out of acceptable range. Packet
dropped!\nPress any key to continue..");
           getch();
           return 1;
    }
    else
    {
          clock_t begin, end;
          clock_t begin_sha,end_sha;
          clock_t begin_rc5,end_rc5;
          clock_t begin_file,end_file;
          clock_t begin_hmac,end_hmac;
          double time_spent;
          double time_spent_sha=0;
          double time_spent_rc5=0;
          double time_spent_file=0;
          double time_spent_hmac=0;
          begin = clock();
          int j,i,y,k,l,m,decimal;
          int index = 0;
          char content[400] = {0x00};

          char * buffer = 0;
          FILE * f = fopen ("memread.txt", "rb");

          if (f)
          {
            fseek (f, 0, SEEK_END);
            length = ftell (f);
            fseek (f, 0, SEEK_SET);
            buffer = malloc (length);
            if (buffer)
            {
               fread (buffer, 1, length, f);
            }
            fclose (f);
          }
          unsigned char ibuf[32];
          unsigned char obuf[64];
          char hashString[64];
          char cipherString[64];
          int hexadecimal;
          int file_counter=0;
          int srno= 65535;
          int loop_saver;
          for(i = 0; i<20; i++)
          {
               begin_sha=clock();
               long combo = i + srno+time_stamp_n1;
               sprintf(ibuf, "%d", combo);
               SHA256(ibuf,strlen(ibuf),obuf);
               for (j=0;j<32;j++)
               { sprintf (&hashString[j*2],"%02x", (unsigned
int)obuf[j]);

               }

               end_sha=clock();
               time_spent_sha+= (double)(end_sha - begin_sha);
               int i;
               for (l=0;l<32;l++)
               {
                  loop_saver=l;
                  for (m=0;m<3;m++)
                  {
```

```
          sprintf (&cipherString[m*2],"%02x", (unsigned
           int)hashString[l]);
               if (m ==2)
                    {
                         cipherString[5]=0;

                         l=loop_saver;
                         break;
                    }
               l=l+1;
          }
          y = (int) strlen(cipherString);
          decimal = 0;
          k=0;
          int ch;
          while(y-1 != 0)
          {
               ch = cipherString[k];
               if('0' <= ch && ch <= '9')
               {
                    decimal = decimal * 16;
                    decimal = decimal + (ch - '0');
               }
               else if('A' <= ch && ch <= 'F')
               {
                  decimal = decimal * 16;
                  decimal = decimal + (ch - 'A')+10;
               }
               else if('a' <= ch && ch <= 'f')
               {
                    decimal = decimal * 16;
                    decimal = decimal + (ch - 'a')+10;
               }
               else
               {
                    decimal=0;
                    break;
               }
               y--;
               k++;
          }
          file_counter+=1;
       int found = 0;
          char chartest[300] = {0x00};
          if(decimal<125000)
          {
               chartest[0] = buffer[decimal];
               if (isalnum(*chartest) || *chartest == '_')
               {
                 content[index++] = buffer[decimal];
          }
          }
          if(found == 0)
            printf("");
     }

}
int h;
begin_hmac=clock();
char key2[] = "012345678";
unsigned char* digest;
```

```
            char mdString[40];
            sprintf(content,"%s%d",content,snsr1_reading_i);
            digest = HMAC(EVP_sha1(), key2, strlen(key2), (unsigned
             char*)content, strlen(content), NULL, NULL);

            for(h = 0; h < 20; h++)
            {
                 sprintf (&mdString[h*2], "%02x", (unsigned
                  int)digest[h]);
            }
            int ret=strncmp(mdString,msg_digest,40);
                 if(ret==0)
            {
                 printf("\n\nDigest match.\n");
            }
            else
            {
              printf("\n\nDigest does not match. Packet dropped.\nPress
                      any key to continue..\n");
          }
                printf("\nmdString: %s",mdString);

            end_hmac=clock();
            time_spent_hmac+= (double)(end_hmac - begin_hmac);

            end = clock();
            time_spent = (double)(end - begin);
            double exectime = time_spent/CLOCKS_PER_SEC;
            long t1 = GetTimeMs64();
            char * f_result=0;

             printf("\nThe Execution Time in nano seconds is %d ",t1-t0);

      }
        getche();
    }
```

Appendix-J

MS VISUAL STUDIO CODE FOR FILE GENERATION

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace MemreadConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {

                FileStream fs = new FileStream("memread128.txt",
FileMode.CreateNew);
                fs.Close();

            }
            catch { }

            try
            {
                string content =
File.ReadAllText(@"C:\Users\IMRAM\Documents\Visual Studio
2008\Projects\memread128\memread128\bin\Debug\main.srec");
                File.WriteAllText(@"C:\Users\IMRAM\Documents\Visual
Studio 2008\Projects\memread128\memread128\bin\Debug\memread128.txt",
content);
            }
            catch { }
            try
            {
                string content =
File.ReadAllText(@"C:\Users\IMRAM\Documents\Visual Studio
2008\Projects\memread128\memread128\bin\Debug\main.ihex");
                File.AppendAllText(@"C:\Users\IMRAM\Documents\Visual
Studio 2008\Projects\memread128\memread128\bin\Debug\memread128.txt",
content);

            }
            catch (Exception ex) { ex.Message.ToString(); }
            File.AppendAllText(@"C:\Users\IMRAM\Documents\Visual Studio
2008\Projects\memread128\memread128\bin\Debug\memread128.txt",
DateTime.Now.ToString());
            FileInfo f = new FileInfo(@"C:\Users\IMRAM\Documents\Visual
Studio 2008\Projects\memread128\memread128\bin\Debug\memread128.txt");
            long s1 = f.Length;
            Console.Write(s1.ToString());
            while (s1 <= 128000)
            {
                Console.WriteLine("The size of file is bytes: {0}",
f.Length);
                File.AppendAllText(@"C:\Users\IMRAM\Documents\Visual
Studio 2008\Projects\memread128\memread128\bin\Debug\memread128.txt",
DateTime.Now.ToString());
                f = new FileInfo(@"C:\Users\IMRAM\Documents\Visual
Studio 2008\Projects\memread128\memread128\bin\Debug\memread128.txt");
                s1 = f.Length;
            }
        }
    }
}
```

Appendix-K

CODE COMPOSER CODE FOR OWCAP using RC-5

```
#include<stdio.h>
#include<time.h>
#include<string.h>
typedefunsignedlongint u32;      /*Should be 32 bit = 4 bytes*/
typedefunsignedchar u16;
#include<stdio.h>

#define uchar unsignedchar // 8-bit byte
#define uint unsignedint // 32-bit word

// DBL_INT_ADD treats two unsigned int a and b as one 64-bit integer
and adds c to it
#define DBL_INT_ADD(a,b,c) if (a > 0xffffffff - (c)) ++b; a += c;
#define ROTLEFT(a,b) (((a) << (b)) | ((a) >> (32-(b))))
#define ROTRIGHT(a,b) (((a) >> (b)) | ((a) << (32-(b))))

#define CH(x,y,z) (((x) & (y)) ^ (~(x) & (z)))
#define MAJ(x,y,z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))
#define EP0(x) (ROTRIGHT(x,2) ^ ROTRIGHT(x,13) ^ ROTRIGHT(x,22))
#define EP1(x) (ROTRIGHT(x,6) ^ ROTRIGHT(x,11) ^ ROTRIGHT(x,25))
#define SIG0(x) (ROTRIGHT(x,7) ^ ROTRIGHT(x,18) ^ ((x) >> 3))
#define SIG1(x) (ROTRIGHT(x,17) ^ ROTRIGHT(x,19) ^ ((x) >> 10))

typedefstruct {
   uchar data[64];
   uint datalen;
   uint bitlen[2];
   uint state[8];
} SHA256_CTX;

uint k[64] = {

0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923
f82a4,0xab1c5ed5,

0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bd
c06a7,0xc19bf174,

0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,0x2de92c6f,0x4a7484aa,0x5cb
0a9dc,0x76f988da,

0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06c
a6351,0x14292967,

0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c
2c92e,0x92722c85,

0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40
e3585,0x106aa070,

0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9
cca4f,0x682e6ff3,
0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef
9a3f7,0xc67178f2
};

typedefstruct {
   uchar data[64];
   uint datalen;
   uint bitlen[2];
   uint state[5];
   uint k[4];
} SHA1_CTX;

voidsha1_transform(SHA1_CTX *ctx, uchar data[])
{
   uint a,b,c,d,e,i,j,t,m[80];

for (i=0,j=0; i < 16; ++i, j += 4)
      m[i] = (data[j] << 24) + (data[j+1] << 16) + (data[j+2] << 8) +
(data[j+3]);
for ( ; i < 80; ++i) {
      m[i] = (m[i-3] ^ m[i-8] ^ m[i-14] ^ m[i-16]);
      m[i] = (m[i] << 1) | (m[i] >> 31);
   }

   a = ctx->state[0];
   b = ctx->state[1];
   c = ctx->state[2];
   d = ctx->state[3];
   e = ctx->state[4];

for (i=0; i < 20; ++i) {
      t = ROTLEFT(a,5) + ((b & c) ^ (~b & d)) + e + ctx->k[0] + m[i];
      e = d;
      d = c;
      c = ROTLEFT(b,30);
      b = a;
      a = t;
   }
for ( ; i < 40; ++i) {
      t = ROTLEFT(a,5) + (b ^ c ^ d) + e + ctx->k[1] + m[i];
      e = d;
      d = c;
      c = ROTLEFT(b,30);
      b = a;
      a = t;
   }
for ( ; i < 60; ++i) {
      t = ROTLEFT(a,5) + ((b & c) ^ (b & d) ^ (c & d))  + e + ctx->k[2]
+ m[i];
      e = d;
      d = c;
      c = ROTLEFT(b,30);
      b = a;
      a = t;
   }
for ( ; i < 80; ++i) {
      t = ROTLEFT(a,5) + (b ^ c ^ d) + e + ctx->k[3] + m[i];
      e = d;
      d = c;
      c = ROTLEFT(b,30);
      b = a;
      a = t;
   }

   ctx->state[0] += a;
   ctx->state[1] += b;
   ctx->state[2] += c;
   ctx->state[3] += d;
   ctx->state[4] += e;
}

voidsha1_init(SHA1_CTX *ctx)
{
   ctx->datalen = 0;
   ctx->bitlen[0] = 0;
   ctx->bitlen[1] = 0;
   ctx->state[0] = 0x67452301;
   ctx->state[1] = 0xEFCDAB89;
   ctx->state[2] = 0x98BADCFE;
   ctx->state[3] = 0x10325476;
   ctx->state[4] = 0xc3d2e1f0;
   ctx->k[0] = 0x5a827999;
   ctx->k[1] = 0x6ed9eba1;
   ctx->k[2] = 0x8f1bbcdc;
   ctx->k[3] = 0xca62c1d6;
}

voidsha1_update(SHA1_CTX *ctx, uchar data[], uint len)
{
   uint t,i;

for (i=0; i < len; ++i) {
      ctx->data[ctx->datalen] = data[i];
      ctx->datalen++;
if (ctx->datalen == 64) {
         sha1_transform(ctx,ctx->data);
         DBL_INT_ADD(ctx->bitlen[0],ctx->bitlen[1],512);
         ctx->datalen = 0;
      }
   }
}

voidsha1_final(SHA1_CTX *ctx, uchar hash[])
{
   uint i;

   i = ctx->datalen;

   // Pad whatever data is left in the buffer.
if (ctx->datalen < 56) {
      ctx->data[i++] = 0x80;
while (i < 56)
         ctx->data[i++] = 0x00;
   }
else {
      ctx->data[i++] = 0x80;
while (i < 64)
         ctx->data[i++] = 0x00;
      sha1_transform(ctx,ctx->data);
memset(ctx->data,0,56);
   }
   // Append to the padding the total message's length in bits and
transform.
   DBL_INT_ADD(ctx->bitlen[0],ctx->bitlen[1],8 * ctx->datalen);
   ctx->data[63] = ctx->bitlen[0];
   ctx->data[62] = ctx->bitlen[0] >> 8;
   ctx->data[61] = ctx->bitlen[0] >> 16;
   ctx->data[60] = ctx->bitlen[0] >> 24;
   ctx->data[59] = ctx->bitlen[1];
   ctx->data[58] = ctx->bitlen[1] >> 8;
   ctx->data[57] = ctx->bitlen[1] >> 16;
   ctx->data[56] = ctx->bitlen[1] >> 24;
   sha1_transform(ctx,ctx->data);

   // Since this implementation uses little endian byte ordering and MD
uses big endian,
   // reverse all the bytes when copying the final state to the output
hash.
for (i=0; i < 4; ++i) {
      hash[i]    = (ctx->state[0] >> (24-i*8)) & 0x000000ff;
      hash[i+4]  = (ctx->state[1] >> (24-i*8)) & 0x000000ff;
      hash[i+8]  = (ctx->state[2] >> (24-i*8)) & 0x000000ff;
      hash[i+12] = (ctx->state[3] >> (24-i*8)) & 0x000000ff;
      hash[i+16] = (ctx->state[4] >> (24-i*8)) & 0x000000ff;
   }
}
voidsha256_transform(SHA256_CTX *ctx, uchar data[])
{
   uint a,b,c,d,e,f,g,h,i,j,t1,t2,m[64];

for (i=0,j=0; i < 16; ++i, j += 4)
      m[i] = (data[j] << 24) | (data[j+1] << 16) | (data[j+2] << 8) |
(data[j+3]);
for ( ; i < 64; ++i)
      m[i] = SIG1(m[i-2]) + m[i-7] + SIG0(m[i-15]) + m[i-16];

   a = ctx->state[0];
   b = ctx->state[1];
   c = ctx->state[2];
   d = ctx->state[3];
   e = ctx->state[4];
   f = ctx->state[5];
   g = ctx->state[6];
   h = ctx->state[7];

for (i = 0; i < 64; ++i) {
      t1 = h + EP1(e) + CH(e,f,g) + k[i] + m[i];
      t2 = EP0(a) + MAJ(a,b,c);
      h = g;
      g = f;
      f = e;
      e = d + t1;
      d = c;
      c = b;
      b = a;
      a = t1 + t2;
   }

   ctx->state[0] += a;
   ctx->state[1] += b;
   ctx->state[2] += c;
   ctx->state[3] += d;
   ctx->state[4] += e;
   ctx->state[5] += f;
   ctx->state[6] += g;
   ctx->state[7] += h;
}

voidsha256_init(SHA256_CTX *ctx)
{
   ctx->datalen = 0;
   ctx->bitlen[0] = 0;
   ctx->bitlen[1] = 0;
   ctx->state[0] = 0x6a09e667;
   ctx->state[1] = 0xbb67ae85;
   ctx->state[2] = 0x3c6ef372;
   ctx->state[3] = 0xa54ff53a;
   ctx->state[4] = 0x510e527f;
   ctx->state[5] = 0x9b05688c;
   ctx->state[6] = 0x1f83d9ab;
   ctx->state[7] = 0x5be0cd19;
}

voidsha256_update(SHA256_CTX *ctx, uchar data[], uint len)
{
   uint t,i;

for (i=0; i < len; ++i) {
      ctx->data[ctx->datalen] = data[i];
      ctx->datalen++;
if (ctx->datalen == 64) {
         sha256_transform(ctx,ctx->data);
         DBL_INT_ADD(ctx->bitlen[0],ctx->bitlen[1],512);
         ctx->datalen = 0;
      }
   }
}

voidsha256_final(SHA256_CTX *ctx, uchar hash[])
{
   uint i;

   i = ctx->datalen;

   // Pad whatever data is left in the buffer.
if (ctx->datalen < 56) {
      ctx->data[i++] = 0x80;
while (i < 56)
         ctx->data[i++] = 0x00;
   }
else {
      ctx->data[i++] = 0x80;
while (i < 64)
         ctx->data[i++] = 0x00;
      sha256_transform(ctx,ctx->data);
memset(ctx->data,0,56);
   }

   // Append to the padding the total message's length in bits and
transform.
   DBL_INT_ADD(ctx->bitlen[0],ctx->bitlen[1],ctx->datalen * 8);
   ctx->data[63] = ctx->bitlen[0];
   ctx->data[62] = ctx->bitlen[0] >> 8;
   ctx->data[61] = ctx->bitlen[0] >> 16;
   ctx->data[60] = ctx->bitlen[0] >> 24;
   ctx->data[59] = ctx->bitlen[1];
   ctx->data[58] = ctx->bitlen[1] >> 8;
   ctx->data[57] = ctx->bitlen[1] >> 16;
   ctx->data[56] = ctx->bitlen[1] >> 24;
   sha256_transform(ctx,ctx->data);

   // Since this implementation uses little endian byte ordering and
SHA uses big endian,
   // reverse all the bytes when copying the final state to the output
hash.
for (i=0; i < 4; ++i) {
      hash[i]    = (ctx->state[0] >> (24-i*8)) & 0x000000ff;
      hash[i+4]  = (ctx->state[1] >> (24-i*8)) & 0x000000ff;
      hash[i+8]  = (ctx->state[2] >> (24-i*8)) & 0x000000ff;
      hash[i+12] = (ctx->state[3] >> (24-i*8)) & 0x000000ff;
      hash[i+16] = (ctx->state[4] >> (24-i*8)) & 0x000000ff;
      hash[i+20] = (ctx->state[5] >> (24-i*8)) & 0x000000ff;
      hash[i+24] = (ctx->state[6] >> (24-i*8)) & 0x000000ff;
      hash[i+28] = (ctx->state[7] >> (24-i*8)) & 0x000000ff;
   }
}

/*Rotation operators. x must be unsigned, to get logical right shift*/
#define ROTL(x,y)  ( ((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))) )
#define ROTR(x,y)  ( ((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))) )
voidRC5_encrypt( u32 *data );
voidRC5_decrypt( u32 *data );
voidkey_setup( unsignedchar *K );
#define w 32   /*u32 size in bits*/
#define r 1    /*number of rounds*/
#define b 16   /*number of bits in key*/
#define c_no 4 /*number of u32 in key, c = max( 1, ceil(8*b/w) )*/
#define t 26   /*size of tables S = 2*(r+1) u32s*/
u32 S[t]; /*expanded key table*/
u32 P = 0xb7e15163, Q = 0x9e3779b9;    /*magic constants*/



voidrc5_encrypt( u16 *data ) /*input pt,ouput ct*/
{
     u32 in[2]={0,0};
     u32 i,A,B;
     in[0] = ((u32)data[0]<<24) ^ ((u32)data[1]<<16) ^
((u32)data[2]<<8) ^ (u32)data[3];
     in[1] = ((u32)data[4]<<24) ^ ((u32)data[5]<<16) ^
((u32)data[6]<<8) ^ (u32)data[7];
     A=in[0]+S[0];
     B=in[1]+S[1];
     for( i=1; i<=r; i++ )
          {
               A = ROTL( A^B, B ) + S[2*i];
               B = ROTL( B^A, A ) + S[2*i+1];
          }
     in[0] = A;
     in[1] = B;

     data[0] = (u16)((in[0]>>24)  &0x000000ff);
     data[1] = (u16)((in[0]>>16)  &0x000000ff);
     data[2] = (u16)((in[0]>>8)   &0x000000ff);
     data[3] = (u16)(in[0]        &0x000000ff);
     data[4] = (u16)((in[1]>>24)  &0x000000ff);
     data[5] = (u16)((in[1]>>16)  &0x000000ff);
     data[6] = (u16)((in[1]>>8)   &0x000000ff);
     data[7] = (u16)(in[1]        &0x000000ff);

}

voidrc5_decrypt( u16 *data ) /*input ct,ouput pt*/
{
     u32 in[2]={0,0};
     u32 i,B,A;
     in[0] = ((u32)data[0]<<24) ^ ((u32)data[1]<<16) ^
((u32)data[2]<<8) ^ (u32)data[3];
     in[1] = ((u32)data[4]<<24) ^ ((u32)data[5]<<16) ^
((u32)data[6]<<8) ^ (u32)data[7];
     B=in[1];
     A=in[0];
     for( i=r; i>0; i-- )
     {
          B = ROTR( B-S[2*i+1], A ) ^ A;
          A = ROTR( A-S[2*i], B ) ^ B;
     }
     in[1] = B - S[1];
     in[0] = A - S[0];

     data[0] = (u16)((in[0]>>24)  &0x000000ff);
     data[1] = (u16)((in[0]>>16)  &0x000000ff);
     data[2] = (u16)((in[0]>>8)   &0x000000ff);
     data[3] = (u16)(in[0]        &0x000000ff);
     data[4] = (u16)((in[1]>>24)  &0x000000ff);
     data[5] = (u16)((in[1]>>16)  &0x000000ff);
     data[6] = (u16)((in[1]>>8)   &0x000000ff);
     data[7] = (u16)(in[1]        &0x000000ff);
}

voidrc5_key_setup( u16 *K )  /*secret input key K[b]*/
{
     u32 i,j,k,u=w/8,A,B,L[c_no];
/*Initialize L, then S, then mix key into S*/
     for( i=b-1,L[c_no-1]=0; i!=-1; i-- )
     {
          L[i/u] = ( L[i/u]<<8 ) + K[i];
     }
     for( S[0]=P,i=1; i<t; i++ )
     {
          S[i] = S[i-1] + Q;
     }
     for( A=B=i=j=k=0; k<3*t; k++,i=(i+1)%t,j=(j+1)%c_no )     /*3*t >
3*c*/
     {
          A = S[i] = ROTL( S[i]+(A+B), 3 );
          B = L[j] = ROTL( L[j]+(A+B), (A+B) );
     }
}

voidmain(void) {
     char  rx_data[100];
          long length;
          char tc;
          FILE * fp ;//= fopen ("out.dat", "rb");
int file_index=0;
          if ((fp=fopen("C:\\out.dat","r"))==NULL){
                        printf("Error! opening file");
                               exit(1);
                           }
                        else
                        {
                             while ((tc = fgetc(fp)) != EOF)
                                 {
                                  rx_data[file_index++] = (char) tc;
                                 }
                        }
          char * msg_digest;
          char  * time_n1;
          char  * snsr1_reading;
     int snsr1_reading_i;
          /* Segregating the information from the received data */
          msg_digest = strtok(rx_data, ",");
          time_n1 = strtok(NULL, ",");
           snsr1_reading = strtok(NULL, ",");

          snsr1_reading = strtok(snsr1_reading, "\n");
          snsr1_reading_i=atoi(snsr1_reading);

     char * buffer;
     char c;
          buffer=(char*)malloc(60000);
               long n=0;
               FILE * f ;//= fopen ("memreadmalpt9.txt", "rb");
               if ((f=fopen("C:\\memreadmalpt9.txt","r"))==NULL){
               printf("Error! opening file");
                      exit(1);
                  }
               else
               {
                    while ((c = fgetc(f)) != EOF)
                        {
                           buffer[n++] = (char) c;
                        }
               }
     time_t now;
     struct tm * ptm;
         now = time(NULL);
         ptm = gmtime ( &now );
     char t_stamp[10];

     char day[2];
     char hour[2];
     char minutes[2];
     char seconds[2];
     if((ptm->tm_mon+1)<10)
         {
     sprintf(t_stamp,"0%d",ptm->tm_mon+1);
         }
     else
         {    sprintf(t_stamp,"%d",ptm->tm_mon+1);}

     if(ptm->tm_mday<10)
            { sprintf(day,"0%d",ptm->tm_mday);}
     else
```

```
                {sprintf(day,"%d",ptm->tm_mday);}
                  strcat(t_stamp,day);
        if((ptm->tm_hour+5)<10)
                {sprintf(hour,"0%d",ptm->tm_hour+5);}
        else
                {sprintf(hour,"%d",ptm->tm_hour+5);}
        strcat(t_stamp,hour);
             if(ptm->tm_min<10)
                {sprintf(minutes,"0%d",ptm->tm_min);}
        else
                {sprintf(minutes,"%d",ptm->tm_min);}
        strcat(t_stamp,minutes);
             if(ptm->tm_sec<10)
                {sprintf(seconds,"0%d",ptm->tm_sec);}
        else
                {sprintf(seconds,"%d",ptm->tm_sec);}
        strcat(t_stamp,seconds);// printf("%s",seconds);
        //   printf("\nt_stamp: %s\n",t_stamp);
        unsignedlong time_stamp=atoi(t_stamp);
        unsignedlong time_stamp_n1=atoi(time_n1);
//      printf("current time:%d",time_stamp);
        int time_difference=time_stamp-time_stamp_n1;
        unsignedint sensor_reading=ptm->tm_sec%10+1;
        char s_reading[2];
        sprintf(s_reading,"%d",sensor_reading);
           /* int time_difference;
        int t_difference;
            t_difference=atoi(time_n1);
            time_difference=time_stamp-t_;*/
        if(time_difference<10)
                     {

                        exit(0);
                        //return 1;
                     }
             elseif(snsr1_reading_i<=0 || snsr1_reading_i>10)
                     {

                        exit(0);//return 1;
                     }

        unsignedchar ibuf[32];
        unsignedchar obuf[64];
        char hashString[64];
        char cipherString[64];
        int hexadecimal;
             int file_counter=0;
        int srno= 65535;
        int loop_saver;
            SHA256_CTX ctx;
            SHA1_CTX ctx_sha1;
        int j,i,y,k,l,m,decimal,h;
        int index=0;
        long combo;
        char ch;
        char content[400] = {0x00};
        for(i = 0; i<20; i++)
            {
                    combo =(long) i + srno+time_stamp;
        sprintf(ibuf, "%d", combo);
                    sha256_init(&ctx);
                    sha256_update(&ctx,ibuf,strlen(ibuf));
                    sha256_final(&ctx,obuf);
             for (j=0;j<32;j++)
                    {
                sprintf (&hashString[j*2],"%02x", (unsignedint)obuf[j]);
                    }
                unsignedchar key[b] =
                    {
                        0x10,0x20,0x30,0x40,0x50,0x60,0x70,0x80,
                        0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
                    };
                rc5_key_setup( key );
                rc5_encrypt( hashString );
                for (l=0;l<32;l++)
                        {
                         loop_saver=l;
                        for (m=0;m<3;m++)
                           {
                                sprintf (&cipherString[m*2],"%02x",
(unsignedint)hashString[l]);//writing obuf hex values to hashString,
one byte in each iteration
                                if (m ==2)
// total 2 iterations for m =0 and m=1. Breaking the loop at m=1
                                    {
                                        cipherString[5]=0;

                                        l=loop_saver;
                                        break;
                                    }
                                l=l+1;
                            }
                            y = (int) strlen(cipherString);
                            int decimal = 0;
                            k=0;
                            int ch;
                        }
                while(y != 0)
                            {
                                ch = cipherString[k];
                                if('0' <= ch && ch <= '9')
                                {
                                    decimal = decimal * 16;
                                    decimal = decimal + (ch - '0');
                                }
                                elseif('A' <= ch && ch <= 'Z')
                                {
                                  decimal = decimal * 16;
                                  decimal = decimal + (ch - 7);
                                }
                                elseif('a' <= ch && ch <= 'z')
                                {
                                    decimal = decimal * 16;
                                    decimal = decimal + (ch -
                                                'a')+10;
                                }
                                else
                                {
                                    decimal=0;
                                    break;
                                }
                                y--;
```

```
                              k++;
                         }
          decimal = decimal % 127997;
          int found = 0;
                         char chartest[400] = {0x00}; //chartest
                         if(decimal<128000)
                         {
                              chartest[0] = buffer[decimal];
                              if (isalnum(*chartest) || *chartest ==
'_')

                              {
                                content[index++] = buffer[decimal];
//content is a string which saves the filtered chars
                              }
                         }
         }
          char key2[] = "012345678";

               // The data that we're going to hash using HMAC
          unsignedchar* digest;
          char mdString[40];
          sprintf(content,"%s%d",content,sensor_reading);
          sha1_init(&ctx_sha1);
         sha1_update(&ctx_sha1,content,strlen(content));
         sha1_final(&ctx_sha1,digest);
          //lrad_hmac_sha1((unsigned
char*)content,strlen(content),key2,strlen(key2),digest);
          for(h = 0; h < 20; h++)
          {
               sprintf (&mdString[h*2], "%02x", (unsignedint)digest[h]);
          }
          int ret=strncmp(mdString,msg_digest,40);
          if(ret==0)
                    {
                         printf("\n\nDigest match.\n");
                    }
                    else
                    {
          printf("\n\nDigest does not match. Packet dropped.\nPress any
key to continue..\n");
                  }
          printf("\nmdString: %s,t_stamp: %s,sensor_reading:
%s",mdString,t_stamp,s_reading);
          FILE *fh = fopen("out.dat", "w");
          fprintf(fh,"%s,%s,%s\n",mdString,t_stamp,s_reading);
          close(fh);

          printf("Exit now!");
}
```

BIBLIOGRAPHY

[1] N. Falliere, L. O. Murchu, and E. Chien, “W32.stuxnet dossier,” Symantec, Tech. Rep., Feb. 2011. [Online]. Available:http://www.symantec.com/content/en/us/enterprise /media /securityresponse / whitepapers / w32stuxnet dossier.pdf

[2] A. Matrosov, E. Rodionov, D. Harley, and J. Malcho, “Stuxnet under themicroscope,” eset, Tech. Rep., 2010. [Online]. Available: http://www. eset.com/resources/white-papers/Stuxnet

[3] S. Karnouskos, "Stuxnet worm impact on industrial cyber-physical system security, " in IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society. IEEE, 2011, pp. 4490-4494

[4] Falliere, N.; Murchu, L.O.; Chien, E. W32.Stuxnet Dossier (2012). Available online: http://www.symantec.com/connect/blogs/updated-w32stuxnet-dossier-available

[5] Broad, W.J.; Markoff, J.; Sanger, D.E. Israeli test on worm called crucial in Iran nuclear delay. *The New York Times*, 15 January 2011. Available online: http://www.cfr.org/iran/nyt-israeli-test-worm-called-crucial-iran-nuclear-delay/p23850

[6] R v Boden [2002] QCA 164; Supreme Court of Queensland: Brisbane, Australia, 2002. Available online: http://archive.sclqld.org.au/qjudgment/2002/QCA02-164.pdf (accessed on 11 July 2012)

[7] http://www.mdpi.com/1999-5903/4/3/672/htm#sthash.hiW7a8Wn.dpuf

[8] D. G. Padmavathi, M. Shanmugapriya et al., “A survey of attacks,Security mechanisms and challenges in wireless sensor networks,” arXivPreprint arXiv: 0909.0576, 2009

[9]A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. E. Culler, “Spins:Security protocols for sensor networks,” Wireless networks, vol. 8, no. 5,pp. 521–534, 2002

[10] S. Zhu, S. Setia, and S. Jajodia, “Leap: efficient security mechanismsfor large-scale distributed sensor networks,” in Proceedings of the 10thACM conference on Computer and Communications Security, 2003, pp.62–72

[11] C. Karlof, N. Sastry, and D. Wagner, “Tinysec: a link layer securityarchitecture for wireless sensor networks,” in Proceedings of the 2nd internationalconference on Embedded Networked Sensor Systems. ACM,2004, pp. 162–175

[12] J. R. Douceur. The Sybil attack. In First International Workshop on Peer-to-Peer Systems (IPTPS ’02), Mar. 2002

[13] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, “Minisec: a secure sensornetwork communication architecture,” in 6th International Symposiumon Information Processing in Sensor Networks. IEEE, 2007, pp. 479–488

[14] S. K. Singh, M. Singh, and D. Singh, “A survey on network security andattack defense mechanism for wireless sensor networks,” InternationalJournal of Computer Trends and Technology, vol. 1, no. 2, pp. 9–17,2011

[15] J. Sen, “A survey on wireless sensor network security,” InternationalJournal of Communication Networks and Information Security (IJCNIS), vol. 1, no. 2, pp. 55–78, 2009

[16] J. Newsome, E. Shi, D. Song, and A. Perrig, “The sybil attack in sensornetworks: analysis & defenses,” in Proceedings of the 3rd internationalsymposium on Information Processing in Sensor Networks. ACM, 2004, pp. 259–268

[17] D. Boyle and T. Newe, “Securing wireless sensor networks: securityarchitectures,” Journal of Networks, vol. 3, no. 1, pp. 65–77, 2008

[18] J. R. Douceur, “The sybil attack,” in Peer-to-peer Systems. Springer,2002, pp. 251–260

[19] M. Demirbas and Y. Song, “An rssi-based scheme for sybil attackdetection in wireless sensor networks,” in Proceedings of the 2006International Symposium on World of Wireless, Mobile and MultimediaNetworks. IEEE Computer Society, 2006, pp. 564–570

[20] J. Wang, G. Yang, Y. Sun, and S. Chen, “Sybil attack detection basedon rssi for wireless sensor network,” in International Conference onWireless Communications, Networking and Mobile Computing. IEEE,2007, pp. 2684–2687

[21] S. Lv, X. Wang, X. Zhao, and X. Zhou, “Detecting the sybil attackcooperatively in wireless sensor networks,” in International Conferenceon Computational Intelligence and Security, vol. 1. IEEE, 2008, pp.442–446

[22] A. Seshadri, A. Perrig, L. Van Doorn, and P. Khosla, “Swatt: Softwarebasedattestation for embedded devices,” in Proceedings of IEEE Symposiumon Security and Privacy, 2004, pp. 272–282

[23] Y. Yang, X. Wang, S. Zhu, and G. Cao, “Distributed software-basedattestation for node compromise detection in sensor networks,” in Proceedings of 26th IEEE International Symposium on Reliable DistributedSystems, 2007, pp. 219–230

[24]K. Song, D. Seo, H. Park, H. Lee, and A. Perrig, “Omap: Oneway memory attestation protocol for smart meters,” in Ninth IEEE International Symposium on Parallel and Distributed Processing withApplications Workshops (ISPAW), 2011, pp. 111–118.

[25] (2013) ”time of check to time of use”. [Online]. Available: http://www.sans.Org/ reading room/whitepapers/securecode/tour-tocttous 1049

[26]I. Chen and Y. Wang, “Reliability analysis of wireless sensor networks with distributed code attestation,” IEEE Communications Letters,vol. 16, no. 10, pp. 1640–1643, 2012

[27] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente, “On the difficultyof software-based attestation of embedded devices,” in Proceedingsof the 16th ACM conference on Computer and Communications Security,2009, pp. 400–409

[28] D.Schellekens, B.Wyseur, and B.Preneel, “Remote attestation on legacy operating systems with trusted platform modules,” *Electronic Notes in Theoretical Computer Science*, vol. 197, no. 1, pp. 59–72, 2008

[29] Yee Wei Law, Jeroen Doumen, and Pieter Hartel. 2006. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Trans. Sen. Netw*. 2, 1 (February 2006), pp. 65-93

[30]Nicolas Fournel, Marine Minier, and Stéphane Ubéda. 2007. Survey and benchmark of stream ciphers for wireless sensor networks. In *Proceedings of the 1st IFIP TC6 /WG8.8 /WG11.2 international conference on Information security theory and practices: smart cards, mobile and ubiquitous computing systems* (WISTP'07), Springer-Verlag, Berlin, Heidelberg, 202-214

[31]Javier López, Jianying Zhou, “Wireless Sensor Network Security”, IOS Press, 01-Jan-2008 – Computers, page 144

[32] H.Khurana, M.Hadley, Ning Lu, and D.A.Frincke, “Smartgrid security issues,” *IEEE Security & Privacy*, vol. 8, no. 1, pp. 81–85, 2010

[33]Rick Kennell and Leah H. Jamieson. Establishing the genuinity of remote computer systems. In *Proceedingsof the 11th USENIX Security Symposium*. USENIX, August 2003

[34]NIST SP 800-90A dated January 2012. Available at http:// csrc.nist.gov / publications / PubsSPs.html

[35]2012, “Using SHA512 hash as random number generator for Gambling services”. Available at https://bitcointalk.org/index.php?topic=77206.0

[36] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, J. D. Tygar.SPINS: Security Protocols for Sensor Networks. In Proceedings of the 7th Annual International Conference on Mobile Computing and Networks (MOBICOM), July 2001, pp. 189-199.

[37] http://www.cs.gmu.edu/~setia/cs818/lectures/spins.pdf

[38]http://en.wikipedia.org/wiki/RC5

[39] Chris Karlof, Naveen Sastry, and David Wagner, “TinySec: A Link Layer security Architecture for Wireless Sensor Networks”, *ACMSenSys 2004,* Nov. 3-5, 2004

[40] David Boyle, Thomas Newe. Securing Wireless Sensor Networks: Security Architectures : Journal of networks, vol. 3, no. 1, January 2008

[41] http://en.wikipedia.org/wiki/Skipjack_%28cipher%29

[42] Lander Casado and Philippas Tsigas. ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System

[43] http://www.keylength.com/en/4/#Biblio4

[44] http://en.wikipedia.org/wiki/CBC-MAC

[45] Mark Luk, Ghita Mezzour, Adrian Perrig, Virgil Gligor. MiniSec: A Secure Sensor Network Communication Architecture. IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA

[46] http://en.wikipedia.org/wiki/ZigBee

[47] Sencun Zhu, Sanjeev Setia And Sushil Jajodia, LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks

[48] http://en.wikipedia.org/wiki/Sybil_attack

[49] W.Stalling and L.Brown, *Computer Security*, Pearson Education International, 2008

[50] Makhdoom, I.; Afzal, M.; Rashid, I., "A novel code attestation scheme against Sybil Attack in Wireless Sensor Networks," *Software Engineering Conference (NSEC), 2014 National* , vol., no., pp.1,6, 11-12 Nov. 2014

[51]Ahmed M., Hunag X., Sharma D., “A Taxonomy of Internal Attacks in Wireless Sensor Network”, ICIS 2012, Kuala Lumpur, Malaysia 2012.

[52] Murat Dener,”Security Analysis in Wireless Sensor Networks,” International Journal of Distributed Sensor Networks Volume 2014, Article ID 303501, 9 page http://dx.doi.org/10.1155/2014/303501

[53] Aziz Mohaisen, Joongheon Kim, “The Sybil Attacks and Defenses: A Survey”, Smart Computing Review, vol. 3, no. 6, pp480-489 December 2013 DOI: 10.6029

[54] Mukul Saini1, Kaushal Kumar2 and Kumar Vaibhav Bhatnagar Efficient and Feasible Methods to Detect Sybil Attack in VANET International. Journal of Engineering Research and Technology. ISSN 0974-3154/. Volume 6, Number 4 (2013), pp. 431-440

[55] Pratch Piyawongwisal, Pengye Xia, “Sybil Attack and Defense in P2P Networks”, 2011, available at web.engr.illinois.edu/~pbg/courses/cs538fa11