DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SECURITY AND RELIABILITY IN EMBEDDED SYSTEMS

by
Mehmet Hilal ÖZCANHAN

June, 2011
İZMİR

SECURITY AND RELIABILITY IN EMBEDDED SYSTEMS

A Thesis Submitted to the Graduate School of Natural and Applied Sciences of Dokuz Eylül University In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering

by

Mehmet Hilal ÖZCANHAN

June, 2011

İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **SECURITY AND RELIABILITY IN EMBEDDED SYSTEMS** completed by **MEHMET HİLAL ÖZCANHAN** under supervision of **ASST. PROF. DR. H. ŞEN ÇAKIR** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

[illegible]

Asst. Prof. Dr. H. Şen ÇAKIR

Supervisor

[illegible]	[illegible]
Asst. Prof. Dr. Gökhan DALKILIÇ	Asst. Prof. Dr. Güleser KALAYCI
Thesis Committee Member	Thesis Committee Member
[illegible]	[illegible]
Associate Prof. Dr. Öznur ÖZKASAP	Asst. Prof. Dr. Ulaş BİRANT
Examining Committee Member	Examining Committee Member

Prof.Dr. Mustafa SABUNCU
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank my advisor Assistant Professor Dr. Şen ÇAKIR and thesis tracking committee member Assistant Professor Dr. Gökhan DALKILIÇ for their contributions to this study.

I would also like to thank to Prof. Dr. Hasan HAVITCIOĞLU for making my late Ph. D. venture possible.

The infrastructure of this work is supported by Dokuz Eylul University Scientific Research Projects (Bilimsel Araştırma Projeleri, BAP) Coordination Unit, numbered as 2009.KB.FEN.044.

I would like to dedicate this work to my children and my wife Göksel ÖZCANHAN for their support and patience during the development and writing of the thesis.

Mehmet Hilal ÖZCANHAN

SECURITY AND RELIABILITY IN EMBEDDED SYSTEMS

ABSTRACT

In this thesis, a security solution for the insecure embedded systems is presented. The proposed solution is implemented in a prototype. The proposed steps of the development and implementation of the solution are the recognition of the weaknesses of the targeted networked embedded systems, the study and analysis of present security standards, the simplification of suitable standards for accommodation in embedded systems, the development of the reached standard-subset, and the implementation of the developed solution in a prototype.

The embedded systems targeted in this thesis are the networked embedded systems; which are vulnerable to the same attacks that computers have to face. The computers have abundant resources compared to the embedded systems. Therefore, the weaknesses of embedded systems were studied in detail, in order to be able to devise a solution, which covers most.

A close examination of present standards to gather information on previously proposed solutions for computers was carried out. A standard that provides end-to-end solution to network communication was carefully simplified to a size and complexity that can fit into low capacity embedded systems. This work involves the accurate selection of rules, algorithms and implementation methods from a complex suite of standards.

A model based on the simplifications was developed, first on computers. The developed model was than ported to low capacity, target embedded systems. The developed models were tested for the expected security that was aimed to be provided to the embedded systems.

The developed solution was later implemented on computers first. Then the tested and debugged result was ported onto a prototype embedded system. Having reached the goal of providing security for low capacity embedded systems, the solution was improved to address the security of fixed-configuration embedded systems, where our solution cannot be mounted. This involved the implementation of our model in another prototype embedded system that acts as a protector, when placed in front of the fixed-configuration device.

It was observed that implementing a subset of a standard made for computers in embedded systems is feasible. Instead of running embedded systems insecurely; by relying on the peripheral security of the network, it is far better to use our solution in mission critical applications of the embedded systems.

Keywords: Embedded Systems, Security, Micro-controller, Authentication, Attack, RFC, IPsec, Local Area Network.

GÖMÜLÜ SİSTEMLERDE GÜVENLİK VE GÜVENİLİRLİK

ÖZ

Bu tezde güvenliği olmayan gömülü sistemlere bir güvenlik çözümü sunulmaktadır. Önerilen çözüm bir prototip üzerinde gerçekleştirilmiştir. Geliştirme ve uygulama sürecinin safhaları hedeflenen gömülü sistemlerin güvenlik zaaflarının incelenmesi, mevcut güvenlik standartlarının detaylı analizi, uygun standartların gömülü sistemlere sığacak şekilde kısaltılması, erişilen standart alt kümesinin geliştirilmesi ve geliştirilen çözümün bir prototip üzerinde uygulamasını oluşturmaktadır.

Bu tezde hedeflenen gömülü sistemler bilgisayarların karşılaştıkları saldırıların aynılarına maruz kalan, ağ bağlantılı gömülü sistemlerdir. Bu sebeple, gömülü sistemlerin güvenlik zaafları dikkatlice incelenerek çoğunu kapsayan bir çözümün tasarımı hedeflenmektedir.

Bilgisayarlar için önceden hazırlanan çözümler hakkında bilgilenmek amacıyla, mevcut standartlar dikkatlice incelenmiştir. Ağ iletişiminde uçtan uca çözüm öneren bir standart kısıtlı kapasitesi olan gömülü sistemlere sığabilecek bir kolaylığa ve boyuta indirgenmiştir. Bu çalışma, karmaşık bir standartlar kümesindeki kuralların, algoritmaların ve uygulamaların doğru olarak seçilmesini içermektedir.

Yapılan kısaltmalara dayanan bir model ilk olarak bilgisayarlar üzerinde geliştirilmiştir. Geliştirilen model daha sonra hedeflen düşük kapasiteli gömülü sistemler üzerine aktarılmıştır. Geliştirilen modeller gömülü sistemler için hedeflenen güvenliğin sağlanıp sağlanamadığı konusunda test edilmişlerdir.

Geliştirilen çözüm daha sonra ilk olarak bilgisayarlar üzerinde uygulama yapılmıştır. Hataları ayıklanan ve test edilen sonuç daha sonra prototip bir gömülü sistem üzerine aktarılmıştır. Düşük kapasiteli gömülü sistemlere güvenlik sağlanması hedefine ulaşıldıktan sonra, çözüm güncellenemeyen sabit konfigürasyonlu gömülü sistemleri de kapsamak amacıyla geliştirilmiştir. Bu uğraşı, modelimizin sabit gömülü sistemin bir ön-koruyucusu gibi çalıştırıldığı başka bir prototip gömülü sistemin üzerine uygulanmasını içermiştir.

Çalışma sonunda, bilgisayarlar için yapılmış bir standardın alt kümesinin indirgenerek gömülü sistemlerde uygulaması gerçekleştirilebilmektedir. Gömülü sistemleri güvenlikten yoksun, çevredeki ağ güvenliğine dayanarak çalıştırmak yerine, kritik uygulamalarda çözümümüzü kullanarak çalıştırmak daha yerinde olacaktır.

Anahtar sözcükler: Gömülü Sistem, Güvenlik, Mikrodenetleyici, Onaylama, Saldırı, RFC, IPsec, Yerel Alan Ağı.

CONTENTS

Page

THESIS EXAMINATION RESULT FORM ii
ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZ v

CHAPTER ONE - INTRODUCTION..1

1.1 Overview...1
1.2 Motivation...2
1.3 The Goal of the Thesis...4
1.4 Thesis Organization...4

CHAPTER TWO - EMBEDDED SYSTEM PROPERTIES, HISTORY AND WEAKNESSES..6

2.1 Embedded Systems Overview...6
2.2 History of Embedded Systems 7
2.3 Properties of Embedded Systems 8
2.3.1 Types of Embedded Systems...10
2.3.1.1 Stand alone Embedded Systems 10
2.3.1.2 Real Time Embedded Systems...10
2.3.1.3.Networked Embedded Systems..11
2.3.1.4 Mobile Embedded Systems.. 11
2.3.2 Examples of Embedded Systems 12
2.3.3 Our Focus on Networked Embedded Systems 13
2.4 Weaknesses of Embedded Systems 15

CHAPTER THREE - TYPES OF ATTACKS ON EMBEDDED SYSTEMS AND SECURITY CONSIDERATIONS......19

3.1 Security Constraints in Embedded Systems......20
3.2 Types of Attacks in General......27
3.3 Attacks on Embedded Systems......31
3.4 Techniques for Resisting Attacks......35
3.5 Security and Reliability of Networked Embedded Systems......36

CHAPTER FOUR - NETWORKS AND EMBEDDED SYSTEM NETWORKING......43

4.1 Networking and the Internet Protocol......43
4.2 Networking in Embedded Systems......45
4.3 The Network Layers......45
4.3.1 Communication between Layers......49
4.3.2 IPv4 Packet Headers......51
4.3.3 Internet Control Message Protocol......54
4.3.4 Multicast......54
4.4 IP Security......55
4.4.1 IP Level Security......59
4.4.2 IP Security Standards......60

CHAPTER FIVE - STANDARD BASED IP SECURITY FOR EMBEDDED SYSTEMS......64

5.1 Is IPsec Standard Suitable For Embedded Systems?......64
5.2 The IPsec Standard......66
5.2.1 Bump in the Stack and Bump in the Wire Implementations......67
5.2.2 IP Tunneling in IPsec......69
5.2.3 Internet Key Exchange Version 2 of IPsec......70
5.2.4 Applying the Rules of IPsec IKEv2 to Embedded Systems......71

5.3 Simplifying IPsec to Fit In Embedded Systems 72
5.3.1 Simplification on the Organization of IPsec 73
5.3.2 Simplification of IKE Exchanges and The Payload Packets 75

CHAPTER SIX - DEVELOPMENT AND TESTING PLATFORM OF PROTO-TYPING 97

6.1 Options of a Development Platform and the Choices Made 97
6.2 The Configuration of the Chosen Platform 103
6.3 The Hardware and Software Configuration of the Embedded System 104

CHAPTER SEVEN - LAB WORK AND THE RESULTING PROTOTYPE . 107

7.1 Pre-Development Phase on Microcomputers 110
7.2 Development and Testing on Micro-computers 112
7.3 Development and Testing on Embedded Systems 115
7.4 Testing of the Embedded System Prototype 116
7.5 Testing of the Embedded System Protector Prototype 118

CHAPTER EIGHT - CONCLUSION 120
8.1 Conclusion 120
8.2 Future Work 121

REFERENCES 122

CHAPTER ONE
INTRODUCTION

An embedded system is any device that includes a programmable computer; but itself is not a general-purpose computer, like the personal computer (PC). An embedded system is designed to perform one or a few dedicated functions (Barr & Massa, 2006), often with real-time computing constraints. Embedded systems are embedded into larger systems as part of the system, which has many other hardware and mechanical parts (Heath, 1995). In contrast, a PC is designed to meet a wide range of end-user requirements (Barr & Massa, 2006).

1.1 Overview

Embedded systems are applied computer systems that contain both hardware and software like PCs but are limited than PCs, in almost every respect. In fact, there is no single definition of embedded systems describing all characteristics. Moreover, an "embedded system" is not a strictly definable term, as most systems have some extensibility or programmability. For example; handheld computers share some elements with embedded systems such as the operating systems and microcontrollers which power them; but they allow different applications to be loaded and peripherals to be connected.

Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers. Complexity varies from a single microcontroller chip, to multiple networked units, peripherals mounted inside a large enclosure. Virtually every electronic device designed and manufactured today is an embedded system. In fact, once one starts looking for them, a few dozen embedded systems can be found in homes or in many ubiquitous applications; systems used everywhere which can reach remote services far away. These applications are putting the embedded devices into communication with other systems, via wired or wireless networks.

It should be mentioned that there is a lot of interest in this area among the international academic community. Some examples are University of Massachusetts (ESSGroup, University of Massachusetts Amherst), University of Twente (DIES University of Twente) and UCLA (EmSec, University of California at Los Angeles), which have dedicated groups working on embedded system security (Figure 1.1).

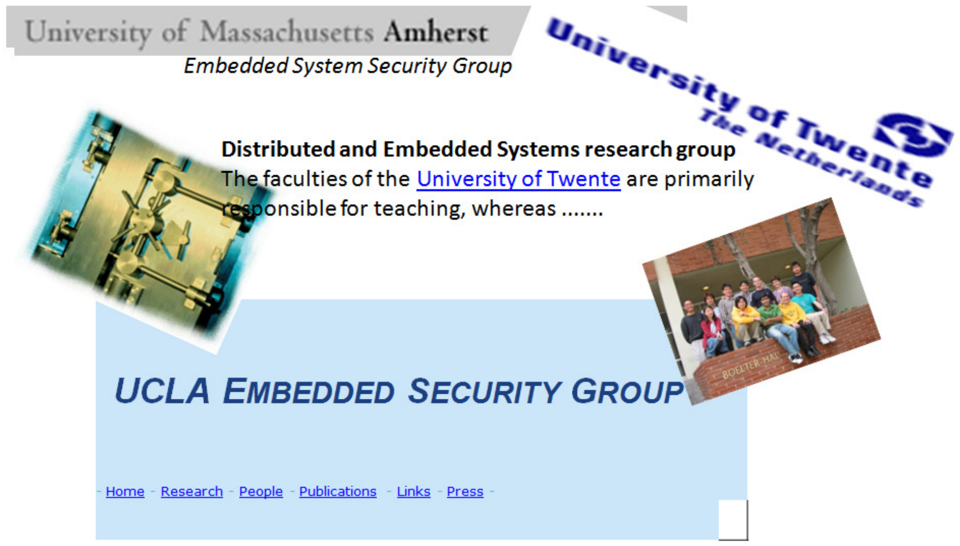

University of Massachusetts Amherst
Embedded System Security Group
University of Twente
The Netherlands
Distributed and Embedded Systems research group
The faculties of the University of Twente are primarily responsible for teaching, whereas ....
UCLA EMBEDDED SECURITY GROUP
Home - Research - People - Publications - Links - Press

Figure 1.1 International Interest in Embedded System Security.

1.2 Motivation

Nowadays, there is a small computer embedded in almost every modern technological instrument. Although these instruments or devices are widely used at home, at work and even in the parks; operating them free from outside interference is still to be put under a general standard. This is an insecure environment for embedded systems and causes a threat to the machine, in which the embedded system resides. News of exploitation of embedded devices is casting suspicion in users' minds. This work is motivated by the fact that a security standard can be reached based on the standards available for computers. The research tries to apply such one standard to embedded systems to form the first step towards a generalized security suite for today's popular devices. An environment based on security standards can increase the sense of security and trust in embedded devices.

The embedded systems are getting widespread in spite of limited resources spared for security. In contradiction, the same attacks launched on computers are targeting the less protected embedded systems. There isn't a day without the news of a new attack reported on embedded devices. To name a few attacks; the credit card thefts, exploited RFID systems, hacked routers (Paul 2009), destroyed factory equipment and even disrupted medical applications (Leyden 2009) can be listed. These circumstances result in the users questioning of the security of embedded devices. Because security in some embedded devices is non-existent and left to the discretion of the larger system housing the embedded device, or is an after-thought in some others.

The skeptic atmosphere created due to the hesitations in whether to perform sensitive operations -like financial transactions or medical operations- through embedded systems or not, results in an unreliable and insecurity feeling among the users. This skepticism has to be removed to increase the trust in embedded systems (Stapko, 2008). To reach this goal security and reliability in embedded systems must be defined. What is understood from these terms, in this research, is summarized here. Security is a generic term used to indicate the confidentiality, integrity, availability and access-control requirements, in embedded systems. Any embedded device lacking any of these requirements is said to lack reliability. For example, unavailability or inconsistent availability of an embedded system roughly means unreliability towards that embedded system.

The fact that the embedded systems are increasingly becoming network enabled aggravates the security issue. With this fact becoming known to everyone, the attacks on embedded systems are increasing (Brodsky, 2009); simply because they are easily reachable through wired or wireless networks.

The gap between the sophisticated attacks prepared for strong computers and the resource limited embedded systems is widened further by the lack of security standards covering embedded systems, in general. Some de facto standards exist which involve specific applications of embedded devices like smartcards, or the mobile devices etc. But that is not enough and there is a need for a new initiative to progress towards a security standard that is be applicable to all of the networked embedded systems.

1.3 The Goal of the Thesis

With restricted resources and many security weaknesses the embedded systems can be attacked in many ways. Each attack is a different challenge and has to be considered at a different security abstraction level. Although, these will be defined later, suffice it to say that only one weakness can be addressed in each research. Our research focuses on one of the most important security areas: Secure communication of embedded systems. As embedded systems are becoming more and more network-enabled (Zurawski, 2006), this issue is of paramount importance and must be addressed first. Starting from this point, the scope of this thesis is defined as security of embedded systems against attacks coming over the network (Stapko, 2008). This puts the focus point on the third layer of the OSI Model; namely the IP Layer. Our goal is to devise a security for all IP driven embedded systems. While doing so, devising a totally new approach would not gain any international support. Therefore, a suitable reference point for standardized embedded system security has to be found. As it will be detailed later, this reference point can come from previously devised standards for computers. Since todays embedded devices are yesterday's computers, in terms of increasing capacity; standards for computers are a good starting point for finding a solution to embedded systems security. At the end, the expectation is that contributions are made to reach the stated goal, in the form of a standard based security feasible in embedded systems and a prototype implemented proving the feasibility of the theoretical proposals.

1.4 Thesis Organization

This thesis is divided into 8 chapters. The motivation of the thesis, a description of the security problem in embedded systems and the goal of securing networked embedded systems are given in Chapter 1. In Chapter 2, detailed information is given about embedded systems. Chapter 3 explains the types of attacks launched on embedded systems and the security issues they raise. Chapter 4 focuses on networking in general and networking issues in embedded systems. In Chapter 5 the feasibility of standard based security, in embedded systems is discussed. Chapter 6 is about the explanation of decisions and strategies made for the choice of the development platform. Chapter 7 gives a detailed account of the lab-work carried out to reach an example prototype. In Chapter 8 there is the conclusion.

CHAPTER TWO
EMBEDDED SYSTEM PROPERTIES, HISTORY AND WEAKNESSES

Embedded systems span to all aspects of modern life and there are many examples of their use. Nowadays, processing has been thoroughly integrated into everyday objects and activities, in the form of ubiquitous systems. This is the most popular area, which employ numerous embedded systems from telephone switches for the network to mobile phones at the end-user. In other words, embedded systems are fully integrated into computer networking and Internet (Zurawski, 2006).

2.1 Embedded Systems Overview

An embedded system is a combination of computer hardware and software and perhaps additional parts, either mechanical or electronic designed to perform a dedicated function. A very good example is the microwave oven. Almost every household has one. The design of an embedded system to perform a dedicated function is in contrast to that of the PC, although it is too comprised of computer hardware, software and mechanical components (Heath, 1995). However, a PC is not designed to perform a specific function, but is able to do many different things. The manufacturer of a PC does not know what the customer will do with it. One may use it for a network file server; another may use it exclusively for playing games. But the embedded system has one, well defined task; guiding a missile for example and nothing else. Since the embedded system is dedicated to specific tasks, design engineers optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced to benefit from mass production.

Frequently, an embedded system is a component within some larger system. For example, modern cars and trucks contain many embedded systems. One embedded system controls the antilock brakes, other monitors the vehicle's emissions and a third displays information related on the dashboard. Some luxury cars have more than 60 embedded processors, including one in each headlight. It is important to point out that a general-purpose computer may interface to numerous embedded systems. For example, a typical computer has a keyboard and mouse, each of which is an embedded system. These peripherals each contain a processor and software and are designed to perform a specific function.

The existence of the processor and software in an embedded system may be unnoticed by a user of the device. Such is the case for the microwave oven. In some cases, it would even be possible to build a functionally equivalent device that does not contain the processor and software. This could be done by replacing the processor-software combination with a custom integrated circuit (IC) that performs the same functions in hardware. Such a processor is called an application specific integrated circuit (ASIC). Some embedded systems consist of only a single IC called a system on chip (SoC). The processor and software combination typically offers more flexibility than a hardwired design, which is the reason why they are preferred in most applications (Zurawski, 2006).

It seems inevitable that the number of embedded systems will continue to increase rapidly. Already there are promising new embedded devices that have enormous market potential. Clearly, manufacturers and individuals who possess the skills and the desire to design the next generation of embedded systems will be in demand for quite some time.

2.2 History of Embedded Systems

One of the first embedded systems was the Apollo Guidance Computer, developed at the MIT Instrumentation Laboratory. The Apollo guidance computer was considered the riskiest item in the project as it employed the newly developed monolithic integrated circuits to reduce the size and weight. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961. It was built from transistor logic and had a hard disk for main memory. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits. This program alone reduced prices of NAND gate ICs from $1000/each to $3/each, permitting their use in commercial products.

From that day on the mass production of microcontrollers, the main building block of embedded systems has lowered the cost to less than a US dollar. This significantly affected the widespread use of microcontrollers in embedded systems. The embedded systems themselves were started to be used in growing number of applications. With the advances in microprocessor technology, the microcontrollers also advanced. The technological trend of reducing the size of the transistor and thus being able to stuff more number of transistors in the same dye area, helped microcontrollers to have growing resources, like increasing ROM and RAM capacities.

This growing capacity of microcontrollers in turn, increased the application areas of embedded systems into diverse fields, like medical and wireless, mobile devices. Using the increased capacity of transistors in application specific circuits and drivers for wireless communication and color displays, exploded the demand for embedded systems. It is true that this demand resulted in the birth of ASICs like digital signal processing (DSP) and graphics chips, instead of the general purpose microcontrollers; nevertheless these ICs are all the building blocks of specialized embedded systems.

The microcontrollers are still increasing their capacity and capabilities in spite of decreasing prices. This is depicted in the fact that more microcontrollers are sold than the microprocessors; nowadays, even though they are also becoming multi-core. The trend is booming the embedded system area further.

2.3 Properties of Embedded Systems

Processors range from simple 4-bit microcontrollers to powerful 128-bit microprocessors specialized DSP or network processors. A diagram showing the relative sizes of an embedded system processor to that of a notebook microprocessor is given in Figure 2.1, for comparison. Notice the big difference in size between an Intel Atom processor used in embedded systems and an Intel Celeron microprocessor used in net books. The processors used in PCs and servers are even larger in size. Some of the products that include these chips run a short assembly program from

ROM with no operating system; many more run real-time operating systems and complex multithreaded C or C++ programs. It's becoming increasingly common to find desktop operating systems based on Linux and Windows operating system; controlling more powerful embedded devices. There are common properties of embedded systems which can be listed as:

- Dedicated function.
- Limited memory, processing power,
- Usually real-time operation.
- Low manufacturing cost.
- Low power consumption.
- Design time subject to tight deadlines by small teams.

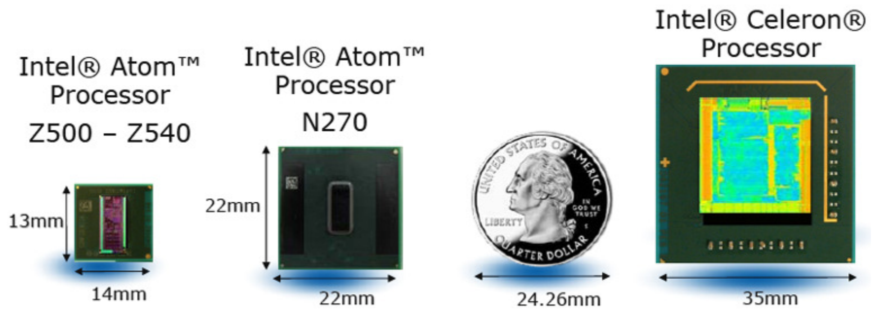


Figure 2.1 Relative Sizes of Processors © Intel Corp.

In addition, the requirements of embedded systems also follow typical characteristics. These are:

- Reliability
- Low Cost
- Low power consumption
- Efficient use of processing power
- Efficient use of memory
- Appropriate execution time

In the previous sections, the difference between PCs and embedded systems was indicated. There are specific properties where embedded systems differ from personal computers (Heath, 1995); one is shown in Figure 2.1. The list of PC properties is:

- Abundant resources,
- Computationally powerful microprocessors,
- High power consumption
- Large memory,
- Big space
- High price

The embedded systems on the other hand have limited computational power, limited primary and secondary memory, usually only one communication channel with the outside world. Nevertheless, of the nine billion processors manufactured in 2005, less than 2% were used in new PCs, Macs, and UNIX workstations. The other 8.8 billion went into embedded systems (Barr, 2006).

2.3.1 Types of Embedded Systems

Embedded systems can be categorized into four mainstream types. These are:

- Stand alone embedded systems.
- Real Time embedded systems
- Networked embedded systems
- Mobile embedded systems

2.3.1.1 Stand alone Embedded Systems

As the name suggests, stand alone systems work by themselves as opposed to networked systems. They just take input and produce output. Usually the response time is not very crucial in standalone systems.

2.3.1.2 Real-Time Embedded Systems

The applications impose different functional requirements onto the operation of embedded systems. Real time embedded systems have to carry out specific functions in a specific amount of time. Response time is very crucial in real time embedded systems. They are used in critical applications where the requirement for real-time operation comes into play. Systems are required to respond within a predefined period of time, mandated by the dynamics of the process under control. Broadly speaking, systems which can tolerate a delay in response are called soft real-time systems; in contrast, hard real-time systems require deterministic responses to avoid changes in the system dynamics which potentially may have negative impact on the process under control, and as a result may lead to economic losses or cause injury to human operators.

2.3.1.3 Networked Embedded Systems

Networked embedded devices are very popular nowadays, especially with the applications using Internet. These systems have emerged in recent years. The complete TCP/IP stack is run to communicate both with each other and controller computers. Some of this type of embedded systems even runs a web server to monitor some parameters and send data over the Internet.

2.3.1.4 Mobile Embedded Systems

Mobile devices are the centre of attraction. With the advent of wireless networks this type of embedded systems have transformed into being part of ubiquitous systems. The reached high data rates have enabled these systems to bring e-mail, World Wide Web, health services and many diverse applications into the pocket of man on the street, bus or taxi. Two such examples are shown in Figure 2.2.

Figure 2.2 Examples of Popular Mobile Devices © Apple

2.3.2 Examples of Embedded Systems

As stated earlier, embedded systems are everywhere in today's world. In fact, the number of embedded systems is more than the number of personal computers, although there are very diverse forms of embedded systems, the most popular embedded devices are found under the following main areas (Mazidi & Mazidi & McKinley, 2006):

- Automotive Industry: Ignition systems, engine control, antilock braking systems, headlights, navigation systems, etc.
- Consumer Electronics: Satellite receivers, DVDs, GPS, ovens, TVs of all types, appliances, cell phones, toys, etc.
- Industrial Control: robotics, control systems, etc.
- Medical Solutions: Infusion pumps, dialysis machines, prosthetic devices, cardiac monitors, etc.
- Networking: Routers, switches, gateways, etc.
- Office Automation: Fax machines, photocopiers, printers, etc.

Consumer electronics include personal digital assistants (PDAs), mp3 players, mobile phones, videogame consoles, digital cameras, DVD players, GPS receivers, and printers. Many household appliances, such as microwave ovens, washing machines and dishwashers, are including embedded systems to provide flexibility, efficiency and features. Advanced air conditioning systems use networked thermostats to more accurately and efficiently control temperature that can change by time of day and season. Home automation uses to control lights, climate, security, audio/visual, surveillance, all of which use embedded devices for sensing and controlling.

Figure 2.3 An MRI scanner contains Embedded Systems

Medical equipment is continuing to advance with more embedded systems for vital signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging devices like X-ray Computed Tomography (CT) and magnetic resonance imaging (MRI) for non-invasive internal inspections. One example of an open MRI scanner is shown in Figure 2.3.

Embedded systems are especially suited for use in transportation and fire safety. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. For fire safety, the systems have ability to handle higher temperatures and continue to operate. In addition to commonly described embedded systems, a new class of miniature wireless devices; called motes, are quickly gaining popularity, in the field of wireless sensors. Wireless sensor networking (WSN), makes use of miniaturization made possible by advanced integrated circuit (IC) design to couple full wireless subsystems to sophisticated sensors; enabling people and companies to measure a myriad of things in the physical world and act on this information through IT monitoring and control systems (Zurawski, 2006). These motes are completely self contained, and will typically run off a battery source for many years before the batteries need to be changed or charged.

2.3.3 Our Focus on Networked Embedded Systems

A networked embedded system application is a collection of spatially and functionally distributed embedded devices, interconnected by a wired or wireless communication infrastructure. Networked embedded systems have communication protocols and they interact with the environment via actuator elements. There is possibly a master node performing control to coordinate computing and communication in order to achieve some pre-defined goals. The emergence of networked embedded systems; frequently termed as distributed embedded systems, has also brought many security issues with it. Especially, now that the distributed systems communicate over the Internet, the danger is even bigger.

Figure 2.4 A Networked Embedded System Example

The benefit of using distributed systems for customers who are constantly on the move is immense, in today's over-populated world. Distributed systems require an evolutionary need to replace the point-to-point wiring connections with network enabled devices. At this point the embedded systems are the same as any network enabled device, which may or may not have a security shield.

The advances in design of embedded systems and falling fabrication costs of semiconductor devices have allowed the infiltration of intelligence into field devices; in the form of sensors and actuators attached to an embedded device. The controllers used with these devices provide on-chip signal conversion, data processing, and communication functions. The increased functionality, processing, and communication capabilities of controllers have been largely instrumental in the emergence of a widespread trend for networked field devices around specialized networks. These are now frequently called ubiquitous systems, as well.

Based on the physical media employed the networks can be divided into three main groups. Namely: wired networks using media such as twisted pair cables (Figure 2.4), fiber optic channels; wireless networks supporting radio frequency (RF) (Figure 2.5) and hybrid networks composed of wired and wireless networks. Although the use of wired field area networks is dominant, the wireless technology offers incentives in a number of application areas, which make them very popular.

The networked embedded systems used in safety-critical applications require a high level of dependability to ensured that a system failure does not lead to a state in which human life, property, or environment are endangered. The dependability issue is critical for technology deployment. One of the main bottlenecks in the development of safety-critical systems is the software development process.

With the growing trend for networking of embedded system and their Internetworking with LAN, Wide Area Network (WAN), and the Internet, many of those systems may become exposed to potential security attacks. For instance, there is a growing demand for remote access to process data at the service centre, which may compromise the integrity of data by transmitting it into air, as in Figure 2.5. The limited resources of embedded nodes pose considerable challenge for the implementation of effective security policies which are resource demanding. These restrictions necessitate a deployment of security mechanisms.

The networked embedded systems appear in a variety of application domains such as, automotive, train, aircraft, office building, industrial monitoring and control, environment monitoring and control, lately in financial applications as well. The need for integration of heterogeneous systems which was difficult in the past because of lacking standards resulted in major initiatives. These had an impact on the integration concepts and the architecture of the protocol stack of networked embedded systems.

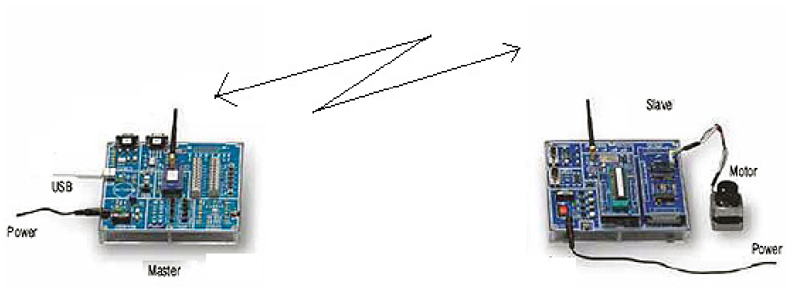


Figure 2.5 A Wireless Networked Embedded System

2.4 Weaknesses of Embedded Systems

As defined before an embedded device in essence any device that has processing power but is not a general purpose computer. This property alone is the one that makes most embedded systems weak towards attacks. The ability to do logical and mathematical operations and carry out functions according to the outcome of these operations is itself the weak point of embedded systems, if not carried out under the control of the owner. Any attacker just needs to fool the embedded system into “producing a foul outcome” to force it carry out a malfunction.

Embedded systems are also classified according to the flexibility of their architecture as either static or extensible architecture (Zurawski, 2006). When static technology is used, the hardware-implemented functions are fixed and inflexible, but they offer higher performance and reduce cost. However, static systems can be more vulnerable to attacks, because once a flaw is identified, it is impossible to patch already deployed systems. This is especially critical in the case of large installations; e.g. SIM cards for cellular telephony or pay-per-view TV. Static systems should be implemented only once and correctly, which is a high expectation in computing.

In contrast, programmable systems are not limited as static ones, but they can be proven flexible in the hands of an attacker, as well. System flexibility may allow an attacker to manipulate the system in ways not expected or defined by the designer. The goal is to add flexibility and programmability to disarm the attacker and to guarantee not to open new exploitation opportunities. Programmability however is typically achieved through the use of specialized software over a general-purpose processor or hardware. Therefore, they do not take into account the reduced memory and computation powers of embedded systems. Thus, these software solutions themselves may have weaknesses.

The increasing capabilities of embedded systems combined with their decreasing cost have enabled their adoption in a wide range of applications and services, from financial, personalized entertainment services to automotive and military applications. In addition to the typical requirements for responsiveness, reliability, availability, robustness, and extensibility, many conventional embedded systems and applications have significant security requirements, as in smartcard applications. However, security is a resource-demanding function that needs special attention in embedded computing. Demanding resources mean higher costs, which are in contrast to the policies of many manufacturers (Zurawski, 2006). Furthermore, the wide deployment of small devices which are used in critical applications has triggered the development of new, strong attacks that exploit more systemic characteristics (Stapko, 2008). In contrast to traditional attacks focused on algorithmic characteristics; due to the ability of attackers to experiment with some of the physical devices used in secure applications, many diverse attacks have come into light.

Thus, design of secure embedded systems requires special attention. But, by definition an embedded system is always a part of another system that carries out one (or a finite number) of the specific functions of the overall system. Therefore, before taking up the security of embedded systems several classifications and categorizations have to be done. First of all the communicating entities must be classified. This allows us to recognize the overall system from scratch whether the embedded system is:

- Independent, singular,
- Connected, singular,
- Connected pair,
- Connected, many linked.

Then there is the classification of the application as being:

- Client/Server
- Equal (Server-Server, Client-Client)
- Controller/Agent

This identifies the role and the communication capabilities of the embedded system against a high capacity controller-computer. This brings out openly whether the embedded system is dependent on other systems, defining a categorization as:

- Standalone Embedded Systems
- Networked Embedded Systems

Another categorization is on the real time property of devices. The embedded system is defined as mission critical if it is a real time embedded system, probably carrying a real time operating system, which may lead to further weaknesses.

The final categorization is about the mobility of the embedded system. If the embedded system is integrated in a mobile system, then it is most probably using wireless communication depending on the application. Even if not communicating at all, it can be taken home and tampered with; which causes yet another weakness. Apart from having the risk of being tampered away from eyes, a mobile embedded system using wireless communication this may end up in another weakness, if the communications are not protected.

As a result, it can be stated that embedded systems have been categorized and each category has many weaknesses to consider. This research focuses on the networked embedded systems, which are deployed to fulfill mission critical applications. The specific weaknesses of networked embedded systems must be paid special attention and provisions must be made to avoid losses to their users. The issue of security in embedded systems is taken up in Chapter 3.

CHAPTER THREE
TYPES OF ATTACKS ON EMBEDDED SYSTEMS AND SECURITY CONSIDERATIONS

It is not surprising to any of us that attacks on computers connected to the Internet are on the rise. Similarly, the attacks on smaller, Internet enabled devices are also on the rise (Paul, 2009). US drone, factory SCADA machines (Brodsky 2009), hospital equipment (Leyden, 2009) have all been attacked. A day does not pass without the news of an attack on the latest popular devices like the X-Box (Huang, 2003), or I-pod or whatever. It appears that the attacks on embedded systems are not towards one particular weakness, but on the contrary - sometimes simultaneously - at different point of weaknesses. Some attacks exploit one weakness to launch a full breach at some other feature. Whatever the attack is, the Dolev-Yao model (Dolev & Yao, 1983) has to be accepted which assumes that an attacker is able to block, modify, eavesdrop on any message or inject his own. This calls for the need to identify the capabilities of an attacker first and then classify the attacks. Only after that the details of the attacks themselves can be analyzed.

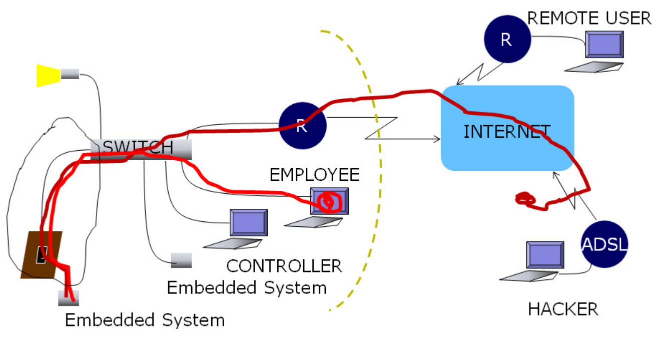


Figure 3.1 Location of attackers and attack routes

The capabilities of the malicious users (attackers) can be classified depending on their knowledge, their hardware–software equipment, and their funds. A classification of the attacker profiles is given below:

1. Class I — clever outsiders: Very intelligent attackers, not well funded and with no sophisticated equipment. They do not have specific knowledge of the attacked

system; basically they are trying to exploit hardware vulnerabilities and software vulnerabilities. This type of users is labeled as "HACKER", in Figure 3.1. They are isolated home users with malicious intent, who surf on the Internet looking for an open door to exploit.

2. Class II—knowledgeable insiders: Attackers with outstanding technical background and education, using highly sophisticated equipment and, often, with inside information for the system under attack; such attackers include former employees who participated in the development cycle of the system. They have the "insider" knowledge of the target system. These are labeled as "EMPLOYEE", in Figure 3.1.

3. Class III—funded organizations: Attackers who are mostly working in teams, and have excellent technical skills and theoretical background. They are well funded, have access to very advanced tools and also have the capability to analyze the system—technically and theoretically—developing highly sophisticated attacks. These are not in Figure 3.1, as they are usually based outside, with capabilities to tap into the network anywhere. This type is the most dangerous class of attackers.

As it can be seen from above, the threat of an attacker in Class II or III is very dangerous. Attacks coming from these sources tend to be devastating if the system is left unprotected. Devastation depends on the end of the line, where the embedded system is connected. It can be a bank or a patient and the pain of the attack depends on the loss caused by the attack.

3.1 Security Constraints in Embedded Systems

Security is a generic term used to indicate several different requirements in computing systems. Instead of agreeing on a formal security definition, the community decided to agree on the properties needed to be satisfied, in order to be able to call a system "secure". Depending on the system and its use, several security properties need to be satisfied in each system and operational environment. Overall, secure systems have to meet all or a subset of the following requirements (Zurawski, 2006), (Stapko, 2008):

1. Confidentiality: Data stored in the system or transmitted from the system have to be protected from disclosure; this is usually achieved through data encryption.

2. Integrity: A mechanism to ensure that data received in a data communication was indeed the data transmitted.

3. Nonrepudiation: A mechanism to ensure that all entities (systems or applications) participating in a transaction cannot deny their actions in the transaction.

4. Availability: The system's ability to perform its primary functions and serve its legitimate users without any disruption, under all conditions, including possible malicious attacks that target to disrupt service, such as the well-known Denial of Service (DoS) attacks.

5. Authentication: The ability of the receiver of a message to identify the message sender.

6. Access control: The ability to ensure that only legal users may take part in a transaction and have access to system resources. To be effective, access control is typically used in conjunction with authentication.

These requirements are placed by different parties involved in the development and use of computing systems, like vendors, application providers, and users. For example, vendors need to ensure the protection of their intellectual property that is embedded in the system, that the application programs are not copied to be reproduced illegally.

Meanwhile, end users want to be certain that the system will provide secure user identification such that only authorized users may access the system and its applications, even if the system gets in the hands of malicious users. The system is also expected to have high availability, that is, it will be available under all circumstances. (Ravi et al., 2004) have identified the participating parties in system and application development and use as well as their security requirements. This classification enables us to identify several possible malicious users, depending on a party's view; for example, for the hardware manufacturer, even a legal end user of a portable device (e.g., a PDA or a mobile phone) can be a possible malicious user.

Considering the security requirements and the interested parties above, the design of a secure system requires identification and definition of the following parameters:

1. The abilities of the attackers,
2. The level at which security should be implemented,
3. Implementation technology and operational environment.

The abilities of attackers have been identified in the previous section. The design of secure systems requires special considerations; because security functions are resource demanding, especially in terms of processing power and energy consumption. Implementing security requirements is relatively much easier in microcomputers, which have abundant resources compared to embedded systems. However; the limited resources of embedded systems require novel design approaches in order to deal with trade-offs between efficiency -speed and cost- and effectiveness- satisfaction of the functional and operational requirements-. Therefore, the implementation of both software and hardware security features of embedded systems is a more difficult task as explained in detail below.

The task of providing security in embedded systems is even more challenging when low power constraints exist. Embedded systems are often battery powered, that is, they are power constrained. Battery capacity constitutes a major bottleneck to processing for security on embedded systems. Unfortunately, improvements in battery capacity do not follow the improvements of increasing performance, complexity, and functionality of the systems they power. (Gunther et al., 2001) report the widening "battery gap," is due to the exponential growth of power requirements and the linear growth in energy density. Thus, the power subsystem of embedded systems is a weak point of system security. A malicious attacker, for example, may form a DoS attack by draining the system's battery quicker than usual. (Martin et al., 2004) describe three ways in which such an attack may take place:

1. Service request power attacks,
2. Benign power attacks,
3. Malignant power attacks.

In service request attacks, a malicious user may request repeatedly from the device to serve a power hungry application, even if the application is not supported by the device. In benign power attacks, the legitimate user is forced to execute an application with high-power requirements, while in malignant power attacks malicious users modify the executable code of an existing application, in order to drain as much battery power as possible without changing the application functionality. They conclude that such attacks may reduce battery life by one to two orders of magnitude.

Inclusion of security functions in an embedded system places extra requirements on power consumption due to: (1) extra processing power necessary to perform various security functions, such as authentication, encryption, decryption, signing, and data verification, (2) transmission of security-related data between various entities, if the system is distributed, that is, a wireless sensor network, and (3) energy required to store security-related parameters.

Performance usually dictates an increased cost, which is not always desirable or possible. Embedded systems are often used to deploy performance-critical functions, which require a lot of processing power. Inclusion of cryptographic algorithms that are used as building blocks in secure embedded design may lead to great consumption of system battery. The energy consumption of the cryptographic algorithms used in security protocols has been analyzed well, for example, by

(Potlapally et al., 2003). They present a general framework that shows asymmetric algorithms having the highest energy cost, symmetric algorithms as the next power-hungry category, and hash algorithms at the bottom. The power required by cryptographic algorithms is significant as measurements indicate (Potlapally et al., 2003). Importantly, in many applications the power consumed by security functions is larger than that used for the applications themselves. For example, (Raghunathan et al., 2003) present the battery gap for a sensor node with an embedded processor, calculating the number of transactions that the node can serve working in secure or insecure mode until system battery runs out. Their results state that working in secure mode consumes the battery in less than half time than when working insecurely.

Security processing places significant additional requirements on the processing power of embedded systems, since conventional architectures are quite limited. The term security processing is used to indicate the portion of the system computational effort that is dedicated to the implementation of the security requirements. Since embedded systems have limited processing power, they cannot cope efficiently with the execution of complex cryptographic algorithms, which are used in the secure design of an embedded system. For example, the generation of a 512-bit key for the RSA public key algorithm requires 3.4 minutes for the PalmIIIx PDA, while encryption using DES takes only 4.9 msec per block, leading to an encryption rate of 13 Kbps (Daswani & Boneh 1999). The adoption of modern embedded systems in high-end systems (servers, firewalls, and routers) with increasing data transmission rates and complex security protocols, such as SSL, make the security processing gap wider and demonstrate that the existing embedded architectures need to be improved, in order to keep up with the increasing computational requirements that are placed by security processing.

Cryptography can protect digital assets provided that the secret keys of the algorithms are stored and accessed in a secure manner. For this, the use of specialized hardware devices to store the secret keys and to implement cryptographic algorithms is preferred over the use of general-purpose computers. However, this also increases the implementation cost and results in reduced flexibility. On the other hand, flexibility is required, because modern cryptographic protocols do not rely on a specific cryptographic algorithm but rather allows the use of a wide range of algorithms for increased security and adaptability to advances on cryptanalysis. For example, both the SSL and IPSec network protocols support numerous cryptographic algorithms to perform the same function, for example, encryption. The protocol enables negotiation of the algorithms to be used, in order to ensure that both parties use the desirable level of protection dictated by their security policies.

Apart from the performance issue, a correct cryptographic implementation requires expertise that is not always available or affordable during the lifecycle of a system. Insecure implementations of theoretically secure algorithms have made their way to headline news quite often in the past. An excellent survey on cryptography implementation faults is provided in (Gutmann, 2002), while (Anderson, 1993) focuses on the causes of cryptographic systems failures in banking applications. A common misunderstanding is the use of random numbers. Pure Linear Feedback Shift Registers (LFSRs) and other pseudorandom number generators produce random-looking sequences that may be sufficient for scientific experiments but can be disastrous for cryptographic algorithms that require some unpredictable random input. On the other hand, the cryptographic community has focused on proving the theoretical security of various cryptographic algorithms and has paid little attention to actual implementations on specific hardware platforms. In fact, many algorithms are designed with portability in mind and efficient implementation on a specific platform meeting specific requirements can be quite tricky. This communication gap between vendors and cryptographers intensifies in the case of embedded systems, which can have many design choices and constraints that are not easily comprehensible.

Vendor tailored versions of standard security protocol suites, such as Secure Sockets Layer (SSL) (Kocher et al 1996) and IP Security Protocol (IPsec) (Kent, 1998), may still not be suitable due to excessive demand for resources. Potential security solutions for this kind of systems depend heavily on the specific device or application domain and Internetworking architecture.

Modern embedded systems are characterized by their ability to operate in different environments, under various conditions. Such an embedded system must be able to achieve different security objectives in every environment; thus, the system must be characterized by significant flexibility and efficient adaptation. For example, consider a PDA with mobile telecommunication capabilities that may operate in a wireless environment (Ravi & Potlapally, 2002) or provide 3G cellular services; different security objectives must be satisfied in each case. Another issue that must be addressed is the implementation of different security requirements at different layers of the protocol architecture. Consider, for example, a mobile PDA that must be able to execute several security protocols, such as IPSec and SSL, depending on its specific application.

Importantly, availability is a significant requirement that needs special support, considering that it should be provided in an evolving world in terms of functionality and increasing system complexity. Conventional embedded systems should target to provide high availability characteristics not only in their expected, attack-free environment but in an emerging hostile environment as well.

Inclusion of security in embedded system design can increase system cost dramatically. The problem originates from the strong resource limitations of embedded systems, through which the system is required to exhibit great performance as well as high level of security while retaining a low cost of implementation.

It is necessary to perform a careful, in-depth analysis of the designed system, in terms of the abilities of the possible adversaries, the environmental conditions under which the system will operate, etc., in order to estimate cost realistically. Consider, for example, the incorporation of a tamper-resistant cryptographic module in an embedded system. As described by (Ravi et al., 2004), a designer can distinguish four levels of security requirements for cryptographic modules. The choice of the security level influences design and implementation cost significantly; so, the manufacturer faces a trade-off between the security requirements that will be implemented and the cost of manufacturing.

Modern secure embedded systems must be able to operate in various environmental conditions, without loss of performance and deviation from their primary goals. In many cases they must survive various physical attacks and have tamper-resistance mechanisms. Tamper resistance is the property that enables systems to prevent the distortion of physical parts. Additionally to tamper-resistance mechanisms, there exist tamper-evidence mechanisms, which allow users or technical staff to identify tampering attacks and take countermeasures. Computer systems are vulnerable to tampering attacks, where malicious users intervene in hardware system parts and compromise them, in order to take advantage of them. Security of many critical systems relies on tamper resistance of smartcards and other embedded processors. (Anderson & Kuhn, 1996) describe various techniques and methods to attack tamper-resistance systems, concluding that tamper-resistance mechanisms need to be extended or reevaluated.

Memory technology may be an additional weakness in system implementation. Typical embedded systems have ROM, RAM and EEPROM memory to store data. EEPROM memory constitutes the vulnerable spot of such systems, because it can be erased with the use of appropriate electrical signaling by malicious users (Anderson & Kuhn, 1996).

A variety of architectures and enhancements in security protocols have been proposed, in order to bridge that gap. (Burke et al., 2000) propose enhancements in the Instruction Set Architecture (ISA) of embedded processors, in order to efficiently calculate various cryptographic primitives, such as permutations, bit rotations, fast substitutions, and modular arithmetic. Another approach is to build dedicated cryptographic embedded coprocessors with their own ISA. The Crypto-maniac coprocessor (Wu et al., 2001) is an example of this approach.

3.2 Types of Attacks in General

The attacks are categorized into two mainstream types: *active* and *passive* attacks (Stallings, 2006). This classification is a pre-requisite for determining the focus and the goal of the search for the particular solution. It would be helpful to briefly explain here what these attacks are and how they affect our scope.

Passive attacks (Stallings, 2006) involve eavesdropping, monitoring and traffic analysis. These attacks are difficult to detect, do not alter any data and do not interfere with the task of the parties. It is however dangerous if the data exposed is sensitive. The focus in our research is on and beyond this type of attack. Our work tries to protect the data and increase the confidentiality that it has not been exposed.

Active attacks (Stallings, 2006) on the other hand involve masquerade, replay, modification of messages, denial of service attacks, etc. These attacks can be deadly for embedded systems which deliver critical services. This type of attacks are our primary concern as mission-critical embedded devices are getting more network connected and open to attackers. Our effort is to decrease the risk of an attack, as much as possible.

Having categorized the attacks broadly into two and studied the capabilities of the attackers, the attacks launched on computers in general can be categorized into the following types:

- Software attacks,
- Hardware (Physical) attacks,
- Side channel attacks,
- Network attacks.

As its name suggests, software attacks try to exploit software weaknesses loaded on the computing devices. It involves the study of the software loaded or the behavior of the software. Usually, it is executed by forcing the current software running to overflow to an unused memory area where the malicious software resides. There is also the "software bypass" attack which forces programs to jump over security intended code. This type of attacks targets the confidentiality, access control and integrity of systems. There is other type of software attacks but those are beyond the interest of this research.

Hardware attacks involve the active interference with the computers or devices. The simplest is "sitting down in front of a server's console where the administrator password has been entered by the operator in charge". The attacker gains unlimited powers in such a situation. This type of attacks is also a danger to the integrity and confidentiality of a whole system. Physical attacks on embedded systems are elaborated on in the next section.

Side channel attacks by definition are attacks that do not necessarily target only the software or the hardware but rather, uses information in one to be used to attack the other. These attacks are collection of techniques that identify and exploit information leaks due to physical activity of the device to attack the software of the device (Kocher et al., 2004). This type of attacks puts integrity, authentication and availability of systems under question, which is also elaborated on in the next section.

The final type of attack is the one coming from the network. The route of these attacks is shown in red, in Figure 3.1. There is a big number of network attacks. The attacks target the integrity, confidentiality of the messages exchanged and put the authentication, access control and availability of systems into danger. They are listed below:

- Denial Of Service (DoS) and Distributed DoS (DDoS),
- Reflection,
- Replay,
- Masquerading: pretending to be another party,
- Spoofing : Forge source address,
- Man-in-the-middle : pass messages through , A <=> X <=> B,
- Oracle : take advantage of unintended encryption and decryption services,
- Type confusion: substitution of a different type of message field.

Denial of Service attacks aim at denying or degrading a legitimate user's access to a service or network resource, or at bringing down the servers offering such services themselves. In a more detailed examination the DoS attacking techniques are classified under two types. The first technique is disabling services by breaking into systems ("hacking"), making use of implementation weaknesses as buffer overrun and deviation from proper protocol execution. The second technique is resource depletion by expensive computations, storage of state information, resource reservations (e.g., bandwidth), and high traffic load.

Generally speaking, these techniques can be applied to protocol processing functions at different layers of the protocol architecture of communication systems. For example, jamming of the wireless communication channel represents the principal attacking technique.

Reflection attack is recording and re-introducing a message to fool the initiator of the communication into thinking that a legitimate opposite exists. This attack may lead to a successful DoS attack. Similarly replay attack is recording and later re-introducing a message or part of the message.

Masquerading attack is pretending to be another party, either the initiator or the responder of a communicating couple. It is a typical active attack.

Spoofing attack is forging a source address. It may prove successful especially when the original source address is turned off. The attacker assumes the network address of the original partner.

A typical active attack is the man-in-the-middle type, where the attacker acts as a transparent go-between the communicating partners. The man-in-the-middle X, acts as an intermediate by passing messages of the communicators, after inspecting or changing them; hence fooling the communicators.

Oracle attack is a very sophisticated attack which involves taking advantage of unintended encryption and decryption “services” built into a security protocol. And finally, type confusion is the substitution of a different type of message field into a loosely designed communication exchange.

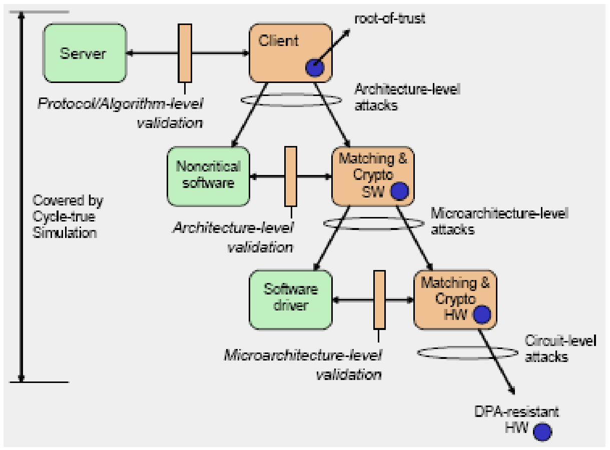
All of the above network attacks are also launched towards the embedded systems for two reasons. Number one, the attacker usually does not know the exact information of the system that is being attacked. In fact it makes no difference to an attacker, as long as there is an open door; i.e. a weak computing device. Number two, the weaker the configuration and resources of the attacked device, the better the chance of success of the attack is. Therefore, the attacker may be happy to know that a weak embedded device has been grounded, leading to a complete network breach. These and other attacks on embedded systems are further elaborated below.

In many cases, embedded systems used for security-critical operations do not implement any tamper resistance mechanisms. Rather, a thin layer of obscurity is preferred, both for simplicity and performance issues. However, as users become more interested in bypassing the security mechanisms of the system, the thin layer of obscurity is easily broken and the cryptographic keys are publicly exposed. The Adobe eBook software encryption (EFF, 2004), the Microsoft X-Box case (Huang, 2003), the USB hardware token devices (Kingpin, 2000), and the DVD CSS copy protection scheme (Touretzky, 2004) are examples of systems that implemented security by obscurity. They were all broken and losses were recorded.

3.3 Attacks on Embedded Systems

Let us start by giving examples to successful attacks on embedded devices. A successful attack on an embedded system at the hardware-software interface is the hacking of the X-Box game console (Huang, 2003). In that system, a secret key used in a software decryption algorithm was probed from the hardware bus during system boot-up. Thus the cryptographic strength of the software algorithm was completely compromised by a loophole at the hardware level, by a physical attack. Other successful attacks are the capture of surveillance images from flying drones by insurgents (Gorman et al, 2009) and the hacking of medical devices (Mocana, 2010) (William & Kohno, 2010).

As it can be observed from the above examples, the attacks on embedded systems are not towards one particular weakness, but on the contrary attacks are made - sometimes simultaneously - at different abstraction levels. The abstraction levels of attacks on an embedded system are shown in Figure 3.2 (Hwang, 2006).


root-of-trust
Server
Client
Protocol/Algorithm-level validation
Architecture-level attacks
Noncritical software
Matching & Crypto SW
Covered by Cycle-true Simulation
Architecture-level validation
Microarchitecture-level attacks
Software driver
Matching & Crypto HW
Microarchitecture-level validation
Circuit-level attacks
DPA-resistant HW
Figure 3.2 Multi-level attacks made on Embedded Systems

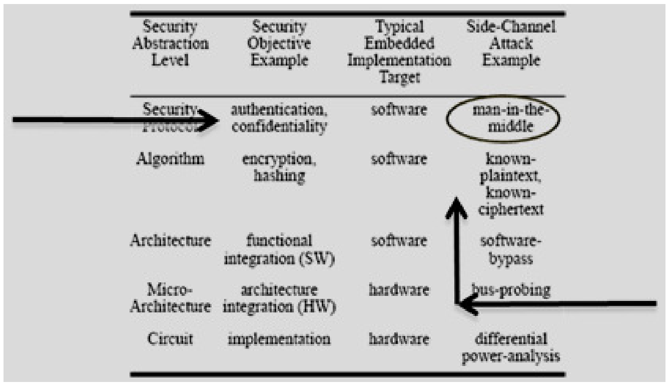
These attacks are protocol/algorithm, architecture, micro architecture and circuit level attacks. A single attack is a brute force or level-specific attack made at the corresponding level. These attack levels can be roughly equated to the general attack types described above. For example the attacks classified as circuit level here, roughly equate to physical attacks above. But physical attacks on embedded systems are inherently easier because those that are mobile can be carried to an obscure place to be poked and those that are small have less complicated circuits than the computers. Micro-architecture-level attacks roughly equate to side-channel and physical attacks, partly combined. Architecture-level attacks mostly involve software attacks complemented with side-channel attacks. Algorithm Level attacks are high level attacks made at the protocol's weakness. They are basically software attacks made on data gathered, exploiting the known weakness of the method to expose the sensitive data.

As an example of a physical attack on embedded systems, fault-induction technique can be given. Devices are always susceptible to erroneous computations or other kinds of faults for several reasons. Faulty computations are a known issue because devices are exposed to radiation that can cause temporary or permanent bit flips, gate destruction, or other problems. Incomplete testing during manufacturing may allow imperfect designs from reaching the market as in the case of device operation in conditions out of their specifications (Quisquater & Samyde, 2001). Careful manipulation of the power supply or the clock oscillator can also cause glitches in code execution by tricking the processor, for example, to execute unknown instructions or bypass a control statement (Kömmerling & Kuhn, 1999).

Some researchers have questioned the feasibility of fault-injection type physical attacks on real systems (Maher, 1997), arguing that fault injection requires expensive and specialized equipment. There have been reports that fault injection can be achieved with low cost and readily available equipment. (Anderson & Kuhn, 1997) and (Anderson,1993) present low-cost attacks for tamper-resistant devices, which achieve extraction of secrets from smartcards and similar devices. (Anderson, 2001) supports the view that the underground community has been using such techniques for quite a long time to break the security of smartcards of pay-TV systems.

There are however; other attacks made on embedded systems that do not necessarily target the level itself but rather, any abstraction level can be used to attack another. These have been defined as side-channel attacks (Kocher, 2004) in Section 3.2. As described before because of their smaller-simpler circuits and mobility attackers are very inventive on side channel attacks on embedded systems. A side channel is any physical channel that can carry information from the operation of a device while implementing a cryptographic operation. Such channels are not captured by the existing abstract mathematical models. The definition is quite broad and the inventiveness of attackers is noticeable. Timing differences, power consumption, electromagnetic emissions, acoustic noise, and faults have been currently exploited for leaking information out of cryptographic systems.

Figure 3.3 shows the type of side-attacks, and the direction of a well-known side attack (Kocher, 2004). The attack goes as follows: The attacker interferes physically with the device and probes the address/data bus. Then the attacker watches the exchange between the user and the device; as a man-in-the-middle, during the authentication phase of the security protocol to obtain an access to the device. Any signals where the user identification and encrypted passwords are exchanged are analyzed. Watching the exact moment of the user-name and password entry; the attacker notes down the instance of the occurrence. On a new device the attacker monitors the exchange not knowing the encrypted information. The data on the bus is copied exactly and replayed later to the device as if typed on a console. If the recording instance was correct the device happily accepts the password, and is accessed. If a fixed pre-shared secret is used in all the devices, then all of them are compromised.



Security Abstraction Level	Security Objective Example	Typical Embedded Implementation Target	Side-Channel Attack Example
Security Protocol	authentication, confidentiality	software	man-in-the-middle
Algorithm	encryption, hashing	software	known-plaintext, known-ciphertext
Architecture	functional integration (SW)	software	software-bypass
Micro-Architecture	architecture integration (HW)	hardware	bus-probing
Circuit	implementation	hardware	differential power-analysis

Figure 3.3 An Example of A Side Attack.

(Kocher, 1996) was the first to present cryptanalysis attacks on implementations of cryptographic algorithms, which were based on the implementation properties of a system. Kocher observed that a cryptographic implementation of the RSA algorithm required varying amounts of time to encrypt a block of data depending on the secret key used. Careful analysis of the timing differences, allowed him to derive the secret key and he extended this method to other algorithms as well (Kocher, 1996). This result came as a surprise, since the RSA algorithm has withstood years of mathematical cryptanalysis and was considered secure. A short time later, Boneh et al. presented theoretical attacks on how to derive the secret keys on implementations of the RSA algorithm, later revised them in (Boneh, 2001).

Embedded systems and especially smartcards are popular targets, for side channel attacks. To understand this, recall that such systems are usually owned by a service provider, such as a mobile phone operator, a TV broadcaster or a bank, and possessed by service clients. The service provider resides on the security of the embedded system in order to prove service usage by the clients, such as phone calls, movie viewing or a purchase, and charge the client accordingly. On the other hand, consumers have the incentive to bypass these mechanisms in order to enjoy free services. Given that SCA are implementation specific and rely, as we will present later, on the ability to interfere, passively or actively with the device implementing a cryptographic algorithm, embedded systems are a further attractive target, given their resource limitation, which makes the attack efforts easier.

The side channels may seem unavoidable and a frightening weakness. However, it should be strongly emphasized that in most cases, reported attacks rely on the detailed knowledge of the platform under attack and the specific implementation of the cryptographic algorithm. Furthermore, the aim of this research is not “to counter all multi layer attacks”. The scope is limited to only attacks coming from the network. Unfortunately, most popular electronic items are networked devices, such as mobile systems.

3.4 Techniques for Resisting Attacks

There are a range of techniques to match the attacks of the hackers. Although each attack requires a special attention, the mainstream precautions can be list as general list of techniques. These are:

- Increasing the key length of an encryption algorithm (Stallings, 2011),
- Tamper proofing (Kocher, 2004) (Ravi et al, 2004),
- Keeping trusted and non-trusted computing parts separated,
- Carefully selecting and designing the crypto algorithm and protocol,
- Thwarting power-analysis attacks. (Kocher, 2004),

- Using specialized circuit styles/gates for a quasi-data-independent power dissipation (Gunther, 2001) ,
- Replacing a macro by two macros, with completely different power consumption profiles,
- Using masking,
- Writing micro-kernels with security in mind from the start (Liedtke, 1996).

Our approach falls into the last category. Our proposal is to insert the network security feature into the very heart of the embedded system. Of course there are many solutions of security offered for wired or wireless network attacks, but almost none of them are based on a standard that is recommended for all embedded systems. Some solution offers have started to appear commercially on the market but these are focused on implementing the same standards recommended for computers in embedded systems too, to the point. Unfortunately this is not viable.

3.5 Security and Reliability of Networked Embedded Systems

Security constitutes a significant requirement in modern embedded computing systems. The widespread use of embedded systems in services that involve sensitive information in contrast to their resource limitations, have led to a significant number of innovative attacks, which result in loss of critical information. Development of secure embedded systems is an emerging field in computer engineering requiring skills from cryptography, communications, hardware, and software. Considering the innovative side-channel attacks that are developed with motivation to break secure embedded systems, new technologies are necessary for countermeasures against known attacks. Clearly, the technical area of secure embedded systems is far from being mature. Innovative attacks and successful countermeasures are continuously emerging, promising an attractive and rich area for research and development.

Secure embedded systems must provide basic security properties, such as data integrity, support for more complex security functions, such as authentication and confidentiality. Furthermore, they have to support the security requirements of applications implemented, using the security mechanisms offered by the system.

Design of secure embedded systems needs to address several issues and parameters ranging from the employed hardware technology to software development methodologies. Although several techniques used in general-purpose systems can be effectively used in embedded system development as well, there are specific design issues that need to be addressed separately. This is because some design issues are unique or weaker in embedded systems, due to the high volume of low-cost methods used for attacks, by malicious users. The major of these design issues are tamper-resistance properties, memory protection, intellectual property protection, management of processing power, communication security, and embedded software design.

Intellectual Property protection of manufacturers is an important issue addressed in secure embedded systems. Complicated systems tend to be partitioned in smaller independent modules leading to module reusability and cost reduction. These modules include intellectual property of the manufacturers, which needs to be protected from third-party users, who sometimes claim and use these modules. The illegal users do not necessarily need to have full, detailed knowledge of the component, since components can be independent modules that very easily be incorporated in a third party system. Fingerprinting, tamper prevention are some techniques for protecting the intellectual property but they require additional processing power, which is limited in embedded systems. The “processing gap” between the computational requirements of security and the available processing power of embedded processors requires special consideration.

Even if the “processing gap” is bridged and security functions are provided, embedded systems are required to support secure communications as well. Often, embedded applications are implemented in a distributed environment where communicating systems exchange sensitive data over a network; e.g. wired or wireless Internet, a Virtual Private Network, the Public Telephone network, etc. In order to fulfill the basic security requirements for secure communications, embedded systems must be able to use strong cryptographic algorithms and to support protocols. One of the fundamental requirements regarding secure protocols is interoperability, leading to the requirement for system flexibility and adaptability. Since an embedded system can operate in several environments, for example, a mobile phone may provide 3G cellular services or connect to a wireless LAN, it is necessary for the system to operate securely in all environments without loss of performance.

Furthermore, as security protocols are developed for various layers of the OSI reference model, embedded systems must be either adaptable to different security requirements at each layer of the architecture or join forces under a standard that protects the majority.

A comprehensive presentation of the evolution of security protocols in wireless communications is provided by (Raghunathan et al., 2003). An important consideration in the development of secure communication subsystems for embedded systems is the limitation of energy, processing and memory resources. The performance/cost trade-off leads to special attention for the design of protocol functions in hardware for high performance; or in software for cost reduction.

Embedded software, such as the operating system or application-specific code, constitutes a crucial factor in secure embedded system design. (Kocher et al., 2004) identify three basic factors that make embedded software development a challenging area: (1) complexity of the system, (2) system extensibility, and (3) connectivity. Embedded systems serve critical, complex, and hard to implement applications, with many parameters that need to be considered, which leads to weak and vulnerable software.

Furthermore, the required extensibility of conventional embedded systems makes the exploitation of vulnerabilities relatively easy. Finally, as modern embedded systems are designed with network connectivity, the higher the connectivity of the system, the higher the risk for a breach, as time goes by. Many attacks can be implemented by malicious users that exploit software glitches and lead to system unavailability, which can have a disastrous impact. As examples, a DoS attack on a military embedded system or a medical device can be given. Common software security faults, such as buffer overflow attacks, heap overflow attacks, array indexing attacks do exist; which are known by many people. Some security programs are available which help designers to analyze the security of their software. Buffer overflow attacks constitute the most widely used type of attacks that lead to unavailability of the attacked system. Malicious users exploit system vulnerabilities and are able to execute malicious code, which can cause loss of sensitive data, a system crash, or prevent legitimate users from using the system.

In the previous paragraphs a review of side channel attacks have been provided. In this part, some of the countermeasures that have been proposed are reviewed. The actual list is long as new results appear continuously, since countermeasures are steadily improving. The proposed countermeasures can be classified into two main classes: hardware protection mechanisms and mathematical protection mechanisms. A first layer of protection is the hardware protection layer, such as layers that do not allow direct access between a malicious user and the system. Or various sensors can be embodied in the device, in order to detect and react to abnormal environmental conditions, such as extreme temperatures, power, and clock variations. Such mechanisms are widely employed in smartcards for financial transactions and other high-risk applications. Such protection layers can be effective against fault-injection attacks, since they shield the device against external manipulation. However, they cannot protect the device from attacks based on external observation, such as power analysis techniques.

The previous countermeasures do not alter the current designs of the circuits, but rather add protection layers on top of them. A second approach is the design of a new generation of chips to implement cryptographic algorithms and to process sensitive information. Such circuits have self-clocking logic and each part of the circuit may be clocked independently (Moore et al., 2002). Fault attacks that rely on external clock manipulation called glitch attacks are not feasible, in this case. Furthermore, timing or power analysis attacks become harder for the attacker, since there is no global clock that correlates the input data and the emitted power. Such countermeasures have the potential to become a common practice. Their application, however, must be carefully evaluated, since they may occupy a large area of the circuit. Such expansions are justified by manufacturers usually in order to increase the system's available memory and not to implement another security feature. Furthermore, such mechanisms require changes in the production line, which is not always feasible.

A third approach targets to implement the cryptographic algorithms so that no sensitive information leaks. Proposed approaches include modifying the algorithm to run in constant time, adding random delays in the execution of the algorithm, randomizing the exact sequence of operations without affecting the final result, and adding dummy operations in the execution of the algorithm. These countermeasures can defeat timing attacks, but careful design must be employed to defeat power analysis attacks too. For example, dummy operations or random delays are easily distinguishable in a power trace, since they tend to consume less power than ordinary cryptographic operations. Furthermore, differences in power traces between profiles of known operations can also reveal permutation of operations. For example, a modular multiplication is known to consume more power than a simple addition, so if the execution order is interchanged, they will still be identifiable.

Insertion of random delays or other forms of noise is another used technique but should be considered carefully, because a large mean value of delay translates directly to reduced performance, which is not always acceptable.

The second class of countermeasures focuses on the mathematical strengthening of the algorithms against such attacks. For example, the RSA blinding technique by (Shamir, 1999) guards the system from leaking meaningful information, because the leaked information is related to the random number used for blinding instead of the key. Thus, even if the attacker manages to reveal a number, this will be the random number and not the key. It should be noted, however, that a different random number is used for each signing or encryption operation. Thus, the faults injected in the system will be applied on a different, random number every time and the collected information is useless.

At somewhere between mathematical and implementation protection, it is good to check cryptographic operations for correctness, to resist fault-injection attacks. However, these checks can be also exploited as side channels of information or can degrade performance significantly. For example, double computations and comparison of the results halves the throughput. Furthermore, in the absence of countermeasures, the comparison function can be bypassed or a fault injection in the comparison function. If multiple checks are employed, measuring the rejection time can reveal the stage of the algorithm the error occurred. If the checks are independent, this can be utilized to extract the secret key, even when the implementation does not output the faulty computation (Sakurai & Takagi, 2003).

Many applications that involve embedded systems are implemented through distributed, networked platforms, resulting in a power overhead due to communication between the various nodes of the system (Wolf, 2000). Considering a wireless sensor network, which is a typical distributed embedded system, one can easily see that significant energy is consumed in communication between various nodes. Factors such as modulation type, data rate, transmit power, and security overhead affect power consumption significantly (Raghunathan et al., 2002). (Savvides et al., 2001) showed that the radio communication between nodes consumes 50 to 60% of the total power. Furthermore, in a wireless sensor network, the security functions consume energy due to key exchange and authentication. Research shows that Diffie–Hellman, implemented using elliptic curve public key cryptography, consumes 1213.7 mJ for 128-bit, 4296 mJ for 192-bit, and 9378.3 mJ 256-bit keys, respectively (Carman et al., 2000).

Although the reliability of embedded systems have been defined slightly differently in the literature, they all lead to the common conclusion that it is the trust to carry out critical tasks through embedded systems, rather than secure computers. If users prefer not to use their mobile embedded system for their banking applications, one cannot talk about the reliability of embedded systems. Hence, there is a need for the existence of an honored security standard being used for the security of embedded systems. Such information would definitely ease the users to feel reliability towards embedded systems. If the security measure guarantees to close down the application in case of any attempt to breach the system, this action would minimize the extend of losses which in turn would increase the reliability toward the used device. As users expect full availability of the device at all times, the knowledge of the existence of a strong authentication security measure, would also increase the reliability on an embedded device that is reached. Because it would mean that if the device is there, it has been authenticated and thus can be trusted. And this is what this work is trying to reach.

Finally, as concluding remarks the attention is drawn to the fact that embedded systems change their operational security characteristics through time. Considering evolving security requirements and improving security protocols, embedded systems need to be flexible and adaptable to changes in security requirements, without losing their performance and availability goals. In this research, the confidentiality, integrity and availability of embedded systems are pursued against attacks coming over the network. Physical and side-channel attacks are not considered here, as they are more hardware related topics.

CHAPTER FOUR
NETWORKS AND EMBEDDED SYSTEM NETWORKING

The computers and embedded devices are constantly in communication with each other. Today almost all commercial applications tend to report back to a server or read some information from it. Where else does the information flow over, but the network? By the general term "network", what physically meant is a connection to a local network, which in turn is linked to the biggest network of all; that is the Internet. Therefore, by a de-facto standard every network is expected to consist of computers and devices linked to each other in the office, which altogether have access to the Internet via a gate-way; also called a router. Some devices may be linked to the router through wireless technology, locally. From the router on the traditional cabled Internet backbone is used. Some wireless devices like cellular phones, reach directly to the Internet via the private network of their Internet service providers (ISP).

This chapter gives general information on the formal and de-facto standards governing networking, among computers. It differentiates the environment in embedded systems than those of computers, by elaborating on specific issues of embedded system networking.

4.1 Networking and the Internet Protocol

Wireless or wired there is another de-facto "standard-duo" used for communication over any network; and that is the Ethernet-Internet Protocol (IP) duo. The Ethernet standard is the de-facto hardware foundation of today's both large and small enterprises; plus home networks. IP is the de-facto protocol used in linking the higher layers of applications which communicate directly with each other, transparently to the user.

Having introduced the basic vocabulary of networking, it is traditional to give the definition and the history of Internet. The modern Internet was born in the late '60s under the name ARPANET. The ARPANET was a research tool for those doing work for the United States government under the direction of the Advanced Research Projects Agency (ARPA). The original contract was awarded to BBN of Cambridge, Massachusetts. In the early 1960s, DARPA (Defense Advanced Research Project Agency) funded a project that connected universities and research agencies through a network called ARPANET. In 1983, the first protocols governing the way computers should communicate were drafted as IP protocols and they replaced the original ARPANET NCP (Network Control Protocols). Later, it was named as the Transmission Control Protocol/Internet Protocol (TCP/IP) which made communication over the network possible by defining a standard of communication was open, simple, and easy to use. This network has grown considerably into what is called "Internet." The Internet is a collection of networks running TCP/IP protocol suite; that is it is a collection of standards. In the '80s there were other networking protocol architectures—ISOs OSI, IBMs SNA, and Digital's DECNET to name a few. However, none of these protocols were as simple and open as the TCP/IP protocol suite. This led to wide deployment, development, and support for the TCP/IP protocol suite.

Briefly, the famous TCP/IP protocol architecture of the Internet consists of various components, which are explained in detail, later in the chapter. These components are:

- Protocol stack: This comprises various layers that communicate among themselves to efficiently transmit the unit of data called a packet. It is called a stack because the layers are stacked on top of each other.
- Addressing: The capability to uniquely identify a destination. In order to communicate with a global entity, it is necessary to uniquely identify the entity. Otherwise, there would be a identity confusion over the whole world.
- Routing: The capability to efficiently determine the path a particular packet is to traverse to reach a destination. Hence, a device performing this task is called a router.

4.2 Networking in Embedded Systems

Networking principles are the same in embedded systems and cannot be any different. But the networking standards are not implemented to the point because of insufficient resources or the cost of accommodating full standards (Zurawski, 2006). The difference comes in the simplifications and omissions made, to fit the protocol stack into the limited memory space and make processing simpler. Because of these missing aspects, usually the security additions to the systems cannot be made; not even during the whole lifetime of the embedded system. Hence, the system is left totally unprotected against attacks coming over the network, whether from local or remote sources (Stapko, 2008).

This is the exact point where this work is focused. It is our argument that rather than leaving the system as security crippled -in terms of adding future protection mechanisms- it would be vital to include a rudimentary security feature based on sound standard, for making the embedded systems safer. Such an attitude is bound to increase the trust of the users towards embedded systems, creating an atmosphere of porting more critical applications on safer devices. While doing so; instead of following a totally new approach, it would be better to consider a subset of the rules, accepted for computers. Such a move would gain international support, since the approach is going to be based on standards which have proved themselves as they are in use for quite some time. Also the idea of a subset standard that can be accommodated in embedded systems can be attractive to manufacturers as well.

Next the definitions and existing standards on networking in general is reminded. It has to be not forgotten that these standards have been designed for fully fledged computers and active network devices with abundant resources in terms of processing power, communication speeds and memory space.

4.3 The Network Layers

The TCP/IP protocol consists of 4 layers as shown in Figure 4.1. Each layer in the protocol stack has well-defined functions and capabilities. Each layer exports defined interfaces so that the layers above and below can use to communicate with it. The layered architecture has many advantages. It simplifies the design and usage of the protocol stack. The design is simplified as each layer interacts only with the layer immediately above and below it. Once the service and the interfaces provided by the layer are identified, each layer can be designed independently. The usage is simplified, as the complexities of the networking stack are hidden from the applications using the stack.

The functionality of each layer is described below. The protocols that implement these services are described later. Further, detailed information can be found in (Frankel 2001) and (Doraswamy and Harkel 2003)

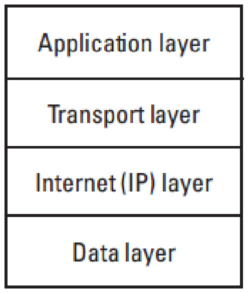


Figure 4.1 Network Layers.

Application Layer: The application layer provides the services for an application to send and receive data over the network. It also provides services such as name resolution (Domain Naming Service, DNS) and applications such as World Wide Web (www) browsers or e-mail. E-mail clients use the services provided by the application layer to communicate with its peers, www servers and e-mail servers respectively. The application layer also defines the interface to the transport layer.

This interface is operating-system dependent. The most popular interface is the socket interface. The socket interface is provided in different software development platforms for the popular operating systems like UNIX and Microsoft. There are many open source implementations. Socket programming is also taught as a course in many undergraduate programs, under the system programming title.

Transport Layer: The transport layer is responsible for providing services to the application layer. In the TCP/IP protocol suite the transport layer provides types of services. The first is connection-oriented or connectionless transport service, where once a connection is established between two applications, the connection stays until one of the applications gives up the connection voluntarily. The application specifies the destination only once, during the establishment of the connection. The best analogy for this is the telephone service. Once a call is established, it stays connected until one speaker disconnects. In connectionless transport, the application has to specify a destination for every single packet it sends. The second type of service is reliable or unreliable transport, where if a packet is lost in the network for some reason (network overload, or node failure), it is retransmitted by the transport layer. The transport layer guarantees the reliable delivery of the packet to the destination. In the unreliable connection, the transport layer does not take up the responsibility of retransmission. It is up to applications to handle cases where a packet does not reach its destination because it was dropped in the network.

An application has to choose the services it requires from the transport layer. There are advantages and disadvantages in choosing different services. In addition, there may be limitations in the combination of services one can choose. Presently, it is invalid to choose connectionless reliable transport as TCP/IP does not implement such a protocol. Applications and their protocols are not familiar to most users but they are essential for the smooth operation of the Internet. Network routing relies on protocols such as the Routing Information Protocol (RIP); the ability to refer to hosts by their names rather than by a lengthy string of numbers results from use of the Domain Naming System (DNS) protocol. Those application protocols rely on the User Datagram Protocol (UDP), a transport protocol that transmits individual packets without checking for loss or duplication. For applications that run over UDP, the applications themselves are responsible for this type of reliability insurance, rather than the underlying transport protocol. The UDP communications model can be compared to the Post Office; messages are sent out and (one hopes) received, but no checking is done to ensure that they actually were received or in what order. Both transport protocols, TCP and UDP, rely on the Internet layer protocol, IP, for the following:

- Transmitting messages from one machine to another;
- Routing the messages so they arrive at the desired destination;
- If the messages are too large to be transmitted by one or more of the network links encountered along the way, breaking the messages into smaller fragments and, at the other end, reassembling the fragments to reconstruct the original message.

The TCP/IP protocol suite implements two protocols at the transport layer. These are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is a connection-oriented protocol ensuring ordered and guaranteed delivery of packets. It has mechanisms built into it to provide these services to the application layer. In addition, TCP also implements mechanisms such as flow control that ensures the destination is not bombarded with packets. UDP is a connectionless protocol that does not ensure either guaranteed or ordered delivery of the packets; nor does it ensure flow control. The choice of using TCP or UDP is entirely up to the application.

The TCP/IP protocol suite identifies the application a packet is destined to by a five tuple: <source address, destination address, source port, destination port, protocol>. This tuple must be unique for each application running on a host. We have already discussed the source and destination address fields. These fields are set in the network header. The source and destination ports are 16-bit fields set in the transport header. The source port is allocated by the source host and the destination is allocated by the destination host. For an application to communicate with another application on another host, it needs to know three things—the address of the destination, the port number on which the application is running, and the protocol over which to communicate. For example, most Web servers are listening on port 80 and use TCP protocol. An application binds to the source and destination port and also specifies the transport protocol to use for the transmission. The transport protocol uses this tuple to identify the application that receives the data.

Internet (IP) Layer: The network layer provides connectionless service. The network layer is responsible for routing packets. Routing can be described as the process that determines the path a packet has to traverse to reach the destination. The devices that decide how to route a packet are called "routers". In order to route the packet, the network layer needs to identify each destination unambiguously. The network layer defines an addressing mechanism. The hosts should conform to the addressing mechanisms to make use of the services offered by the network layer. This is discussed in detail, including the packetized mechanism and the packets headers in the sections, below.

Data Link Layer: The data link layer is responsible for packet transmission on the physical media. The transmission is between two devices that are physically connected. Examples of data-link layers are Ethernet, Token Ring, and Asynchronous Transfer Mode (ATM).

4.3.1 Communication between Layers

As described above, each layer in the protocol stack has a specific function and the layering must be preserved. The application layer cannot talk to the network layer directly. It has to talk through the transport layer.

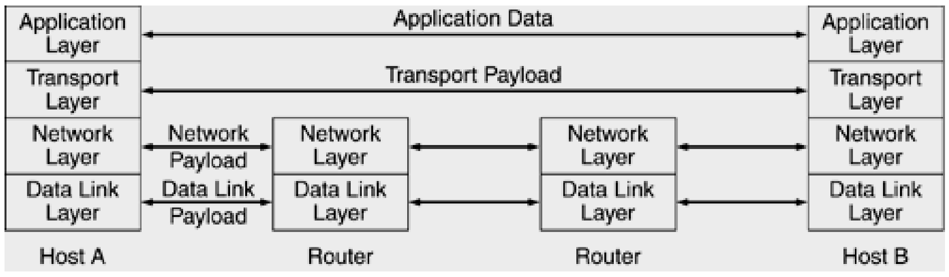


Figure 4.2 Communications between Layers

Figure 4.2 illustrates the layers of a typical system that uses TCP/IP as its networking protocol. When an outbound message is constructed, each layer, from the top to the bottom, inserts its own header in front of the data to be transported and then sends the message to the next (lower) layer for further processing. When an inbound message is received, the process is reversed. Each layer, from the bottom to the top, performs its layer-appropriate processing, strips off its header, and sends the message to the next (upper) layer for further processing. Each layer views a message as having two parts: the layer's header and data. The data in fact generally contains a series of upper-layer headers, followed by the message data destined for the application.

The data flow from source to destination is as shown in Figure 4.3. In both the TCP and UDP headers two fields are present: the source port and the destination port. These two fields are critical in identifying how to process the data once the destination receives it.

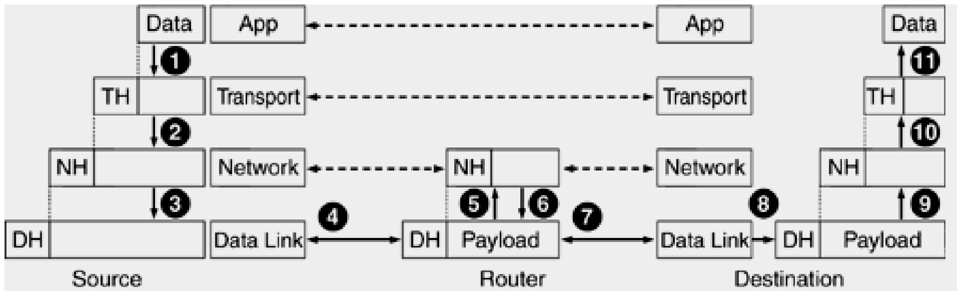


Figure 4.3 Data Flow.

Let us assume that the transport protocol is TCP and the network protocol is IP. An application on the source host sends the data that needs to be transmitted to the destination over the interface to the transport layer. The application identifies the destination it wishes to communicate with. The destination includes the host and an application on the host. The TCP transport layer gets this data and appends a transport TCP header to the payload data, and sends it down to the network layer. The fields in the TCP header help in providing the services requested by the application. The network layer receives the payload from the transport layer which consists of the data and the TCP header. It appends an IP header to this payload. It then sends the payload plus IP header down to the data link layer. In addition, the network layer also identifies the neighbor that the packet needs to be sent to, en route to the destination. The data link layer then appends a data link header to the payload from the network layer. The data link layer identifies the physical address of the next hop the packet should be sent to and sends the packet. The data link layer on the next hop receives the packet, strips the data link header from the packet and sends the packet up to the network layer. The network layer looks at the network header and decides the next hop the packet needs to be sent to en route to the destination and invokes the data link layer. The data link layer appends the data link header to the payload and transmits the packet to the next hop. Procedures 6 and 7 are repeated till the packet reaches the destination.

Upon reaching the destination, the data link layer strips the data link header from the packet and sends it up to the network layer. The network layer then strips the network header from the packet and sends it up to the transport layer. The transport layer then checks the transport header to guarantee that the application is being serviced properly, strips the transport header, identifies the application to which this packet is destined, and sends it up to the application. The application on the destination receives the data that was sent to it by the application on the source.

4.3.2 IPv4 Packet Headers

In the TCP/IP protocol suite, there are two network protocols—IPv4 and IPv6. These protocols are discussed to provide the information for the basis of understanding IP Security. The overwhelming majority of packets that travel over the Internet follow the rules and the format defined by Internet Protocol Version 4 (IPv4). A new protocol, Internet Protocol Version 6 (IPv6), has been also defined and is deployed in limited portions of the Internet. The motivation for the development of IPv6 was the predicted depletion of the IPv4 address space due to the unanticipated increase in the Internet's popularity and use. To limit the complexity of explanations only IPv4 packet header is focused on.

The header shown as the Network Header (NH) which is constructed or processed by the IP layer, referred to as the IP header format is illustrated in Figure 4.4. IPv4 is the most prevalent network layer protocol today. It uses a simple addressing scheme and provides connectionless service. IPv4 has a very mature routing infrastructure. Only those fields that are used in IP security are explained, in detail. For example, the options field is not discussed in detail.

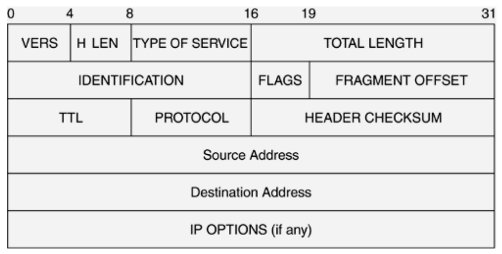


Figure 4.4 IPv4 Header.

Version field (VERS): This 4-bit field is used to indicate the version. This value is 4 for IPv4. The version field is normally used for backward compatibility. When new versions are defined, they may be required to interoperate with the legacy systems.

Header length (H Len): The header length indicates the length of the header in 32 bits (4 bytes). This limits the maximum length of the IPv4 header to 60 bytes. This is one of the limitations of IPv4 that led to the development of IPv6.

Type Of Service (TOS): TOS is used to indicate the traffic requirements of the packet.

Total Length: The length of the datagram in bytes (including the header) in the network byte order. This field indicates the size of the datagram to the network layer at the receiving end.

Identification: The 16-bit identification field is used to uniquely identify an IP datagram. The term IP datagram refers to the transport payload plus IP header, and is used in the context of end hosts. The identification field is used mostly in the context of fragmentation, i.e. when a data is too long to fit into one single packet. The identification field is used to uniquely identify which IP packets belong to an IP datagram.

Flags: Only 2 out of the 3 bits in the flag are defined. The first bit is used to specify not to fragment the IP packet. When this is set, a router sends back a control message to the host indicating its MTU (Maximum Transfer Unit). This is a process by which the end host discovers what size the IP packets it generates should be so that the packets do not get fragmented en route to the destination. This is necessary because fragmentation is detrimental to the operation of the network. The transport layer has to send the entire datagram if a fragment is lost. The second bit is used to indicate if the packet is the last fragment of a fragmented datagram or if there are more to follow. This bit is used in reassembling fragmented packets.

Fragmentation offset: This field indicates the offset of the IP packet in the IP datagram.

Time To Live (TTL): This field is used to avoid packet looping and also to administratively scope the transmission of a packet. The host sets this field to a certain default value and each router along the path decrements this field by 1. If a router sees a packet with a TTL of 1, it drops the packet. This is crucial in case of routing loops as the packet will be roaming in the network forever if nobody drops it.

Protocol: This 8-bit field is used to indicate the transport protocol carried by this IP packet. This field is used by the end host to de-multiplex the packet among various transport protocols.

Header Checksum: The checksum is calculated on the IP header and is used to guarantee the integrity of the IP header. The checksum is not a cryptographic checksum and can be easily forged.

Source address: This 32-bit field indicates the IP address of the source that generated this packet.

Destination address: This 32-bit field indicates the IP address of the destination host.

IP Options: An IP header can optionally carry additional information. As options are not important in understanding IP security, they are not discussed here.

4.3.3 Internet Control Message Protocol

Internet Control Message Protocol (ICMP) is used to ensure the proper operation of the network and for debugging. The protocol runs on top of IPv4 or IPv6. ICMP messages are generated both by hosts and routers to monitor the network and to ensure proper operation of the network. For example, if a router does not have a route to a particular network, it sends an ICMP message back to the host indicating the network is unreachable. If the router drops the packet without any indication, monitoring the network becomes a nightmare. ICMP is used to determine if a host is reachable or not. Also ICMP is used in the determining of maximum transfer unit, for fragmentation of a packet. For example, if a router needs to fragment a packet but the "do not fragment" bit is set, then the router sends back an ICMP message to host indicating the MTU of its link so that the host can generate packets whose size does not exceed this MTU.

4.3.4 Multicast

IP also provides the ability to send a packet to multiple hosts anywhere on the Internet. This is a special case of broadcasting where only interested hosts receive a packet. Consider the example of IP TV, pay-per-view broadcast. If a cable company intends to telecast a program over the Internet to all its subscribers, it has three options. One option is to telecast individually to each subscriber, which has a very high overhead as the same data is duplicated to all the subscribers. It has a lot of undesired side effects. It increases the Internet traffic substantially and also increases tremendously the load on the server that is distributing data. The second option is to telecast by sending a broadcast to the whole Internet. This process is unacceptable because even nonsubscribers get to see the pay-per-view program. Moreover, Internet-level broadcast is very bad. The third option is to telecast only to subscribers by using a technique called multicast. Multicast is an intelligent packet distribution mechanism where only subscribers get the packet. The data is distributed on only those links with subscribers. The traffic is delivered only to those nodes that have subscribed to the traffic.

Multicast packets have the same format as unicast IP packets. However, the destination field in the IP header has a multicast address and not a unicast address. The obvious question that comes to mind is, How do you know where to send the packet if you cannot uniquely identify a host? That is the beauty of multicast. There is support from the routing layer that has the knowledge of the hosts that are listening on a particular multicast address. A detailed discussion of multicast is out of the scope. In fact, multicast discussion is a whole topic in itself.

4.4 IP Security

In today's Internet, there are a lot of protocols designed to secure traffic at various levels described above, in the network. It depends on the security requirements of the application and the user to decide where in the protocol stack, security should be implemented. But, at first irrespective of where in the stack security is implemented, the following basic services have to be provided:

- Key management (Negotiation and storage of keys),
- Confidentiality,
- Nonrepudiation,
- Integrity/authentication,
- Authorization.

Depending on where in the stack the security is implemented, it is possible to provide some or all of the services above. In some cases, it makes sense to provide some capabilities at one layer and other capabilities at a different layer.

Cryptography provides a mechanism that helps in meeting all of the above requirements except nonrepudiation.

However, the embedded systems mostly work in spaces where the computers do not work. This is because some applications have been defined previously and have become standards. Any embedded system wishing to communicate with a computer under one of these standard applications will have to adapt to the pre-defined standards. For non standard applications, the embedded systems use the non-standard space and allowed resource numbers, like unused TCP/IP and UDP/IP ports. Therefore a mechanism that protects all IP applications is necessary since one cannot guess what an embedded system might be using to access the network layer. This layer consideration is shown in Figure 4.5. The attention of this work is focused on IP security, which provides security at the network layer for all applications. A discussion of the advantages and disadvantages of providing security at various layers in the stack is provided below.

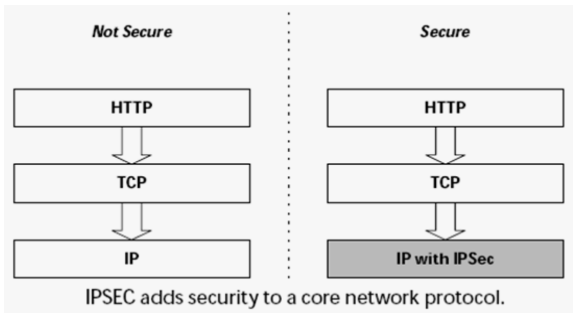


Figure 4.5 IP Layer Security Provided With IPsec

Application-level security has to be implemented in end hosts. Executing in the context of the user enables easy access to user credentials such as private keys, complete access to the data the user wants to protect (which simplifies the task of providing services such as nonrepudiation). An application can be extended without having to depend on the operating system to provide these services. Normally, applications have no control over what gets implemented in the operating system. Application understands the data and can provide appropriate security. The disadvantage of application layer security is that the security mechanisms have to be designed independently for each application. This implies existing applications have to be enhanced to provide security. As each application has to define its own security mechanisms, there is a greater probability of making mistakes and hence opening up security holes for attacks.

In implementing security mechanisms in applications, applications integrate with a system providing the security mechanisms. Examples of such systems are Pretty Good Privacy (PGP), Kerberos, and Secure Shell (SSH). These systems are application-level protocols that provide the capability of key negotiation and other security services. Applications are enhanced to call into this system to use their security mechanisms. One example is the e-mail clients that use PGP to provide e-mail security. Applications should design their own security mechanisms when their needs are specific and they cannot depend on the lower layers to provide those services. One such example is nonrepudiation. It is difficult for lower layer to provide nonrepudiation services as they do not have access to the data.

Providing security at the transport layer has a definite advantage over the application-layer security as it does not mandate enhancements to each application. Existing applications get security services seamlessly. However, obtaining the user context gets complicated. In order to provide user-specific services, assumptions are made that a single user is using the system, which is becoming a popular paradigm. Like application-level security, transport-layer security can only be implemented on an end system. Transport-layer security is protocol specific. Transport Layer Security (TLS) is a protocol that provides security services such as authentication, integrity, and confidentiality on top of TCP. TLS needs to maintain context for a connection and is not currently implemented over UDP as UDP does not maintain any context. As the security mechanism is transport-protocol specific, security services such as key management may be duplicated for each transport protocol.

The World Wide Web currently provides security services using TLS. However, if security services were implemented at the network layer, this can be moved down to the network layer. Another limitation of transport-layer security as it is currently defined is that the applications still need modification to request security services from the transport layer. Implementing security at this layer has many advantages. First off, the overheads of key negotiation decrease considerably. This is because multiple transport protocols and applications can share the key management infrastructure provided by the network layer. Also, if security is implemented at lower layers, fewer applications need changes. It reduces the explosion in the implementation of security protocols at the higher layer. If security is implemented at higher layers, each application has to design its own security mechanism. This is overkill and the probability of someone making a mistake is much higher. Also, security is provided seamlessly for any transport protocol.

Next level where security may be inserted is the network level. One of the most useful features of network layer security is the ability to build VPNs and intranets. Because VPNs and intranets are subnet based, and network layer supports subnet-based security, it is easy to implement VPNs and intranets. The disadvantage of implementing security at the network layer is the difficulty in handling issues such as non-repudiation of data. This is better handled in higher layers. It is more difficult to exercise control on a per user basis on a multiuser machine when security is implemented at network layer. However, mechanisms can be provided to perform user-based security on end hosts. On the routers, there is no context of user and this problem does not arise.

The last level where security can be implemented is the Data Link Layer. If there is a dedicated link between two hosts/routers and all the traffic needs to be encrypted for the fear of snooping, one can use hardware devices for encryption. The advantage of this solution is speed. However, this solution is not scalable and works well only on dedicated links. Moreover, the two entities involved in communication have to be physically connected. This type of model is useful in automatic teller machines where all the machines are connected via dedicated links to a central office. If ATM machines were connected to an IP network instead of dedicated secure links, the data link layer security would not suffice and one would have to move up one layer to provide security services.

4.4.1 IP Level Security

The insecure travelling of unprotected IP packets over the network is shown in Figure 4.6. IP Packets have no inherent security. It is relatively easy to forge the addresses of IP packets, modify the contents of IP packets, replay old packets, and inspect the contents of IP packets in transit. Therefore, there is no guarantee that IP datagrams received are (a) from the claimed sender (the source address in the IP header); (b) that they contain the original data that the sender placed in them; or (c) that the original data was not inspected by a third party while the packet was being sent from source to destination. Therefore, if the IP datagram can be protected by guaranteeing the above three properties, then the datagrams can be classified as secured (Stallings, 2011).

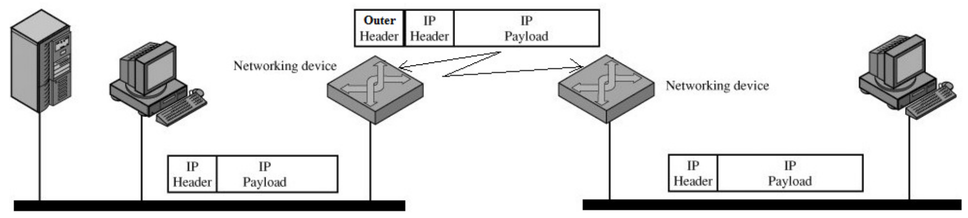


Figure 4.6 Insecure Relay of IP Packets

The secure journey of protected IP packets over the network is shown in Figure 4.7. IP Packets have security, because there is a security header added in front of the packet identifying the secure payload, and the payload is now protected by encryption. It is no longer possible to modify the contents of IP packets and inspect the contents of IP packets in transit. With additional security features it is not possible to replay a packet or forge the IP address. Therefore, the IP datagram can be defined as secure, now (Stallings, 2011).

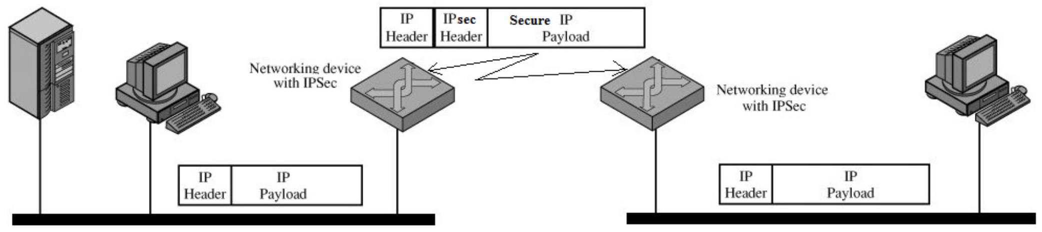


Figure 4.7 Secure Relay of IP Packets

4.4.2 IP Security Standards

There is a standard that aims at securing IP packets or datagrams and it is called IPsec. IPSec is a method of protecting IP datagrams. This protection takes the form of data origin authentication, connectionless data integrity authentication, and data content confidentiality (Doraswamy & Harkins, 2003). In fact, there is no other suit of standards that collectively try to provide security at the IP level that covers all systems. The frame of reference in which IPsec operates is that of the Internet Protocol (IP), as shown in Figure 4.5.

IPsec is the only protocol that can secure all and any kind of Internet traffic. IPSec also allows per flow or per connection security and thus allows for very fine-grained security control. In Table 4.1 the IPsec standard is compared with Application and Transport Layer security protocols. IPsec is only inferior to the other security protocols in two categories, in the case of computers. The first is that it does not have tailored services to particular to every application at higher levels. Both for embedded systems this is not a disadvantage because the applications are not known or standard. All that is expected is security of all packets of all sorts of services. Therefore, this can be a problem for computer applications but not embedded systems. The second is IPsec is not easy to deploy and maintain. Since the controller and the embedded system controlled by the controller are not totally strangers to each other, the IP address and the IP ports that will be used are known to each other. This greatly simplifies deployment issue. The controller and embedded systems are usually used in pairs. Or sometimes a single controller controls many embedded systems; rarely many controllers control only one embedded system. Therefore, since there are usually pairs who know each other's IP address and service ports then deployment is not a disadvantage.

Table 4.1 Comparison of Security Protocols at Different Layers

Protocol arch	Example	Full Security	Multiple App	Tailored Services	Transparent to App	Easy to Deploy
Separate protocol layer	SSL	X	X	O	O	X
Application Layer	S-Http	X	O	X	O	X
Integrated with core	Ipsec	X	X	O	X	O
Paralell Protocol	Kerberos	O	X	O	O	X

IPSec provides a standard, robust, and extensible mechanism in which to provide security to IP and upper-layer protocols (e.g., UDP or TCP). A default, mandatory-to-implement suite of algorithms is defined to assure interoperability between different implementations, and it is relatively straightforward to add new algorithms without breaking interoperability (Frankel, 2001). Hence a secure communication between an embedded system and its controller is shown in Figure 4.8.

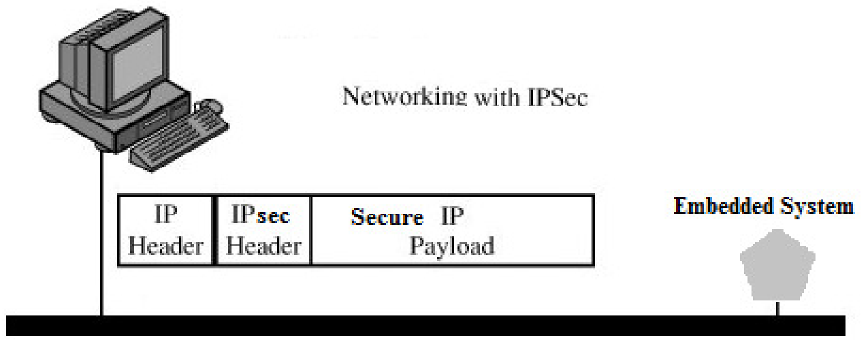


Figure 4.8 Secure Transmission of IP Packets Between An Embedded System and its Controller

IPSec protects IP datagrams by defining a method of specifying the traffic to protect, how that traffic is to be protected, and to whom the traffic is sent. IPSec can protect packets between hosts, between network security gateways (e.g., routers or firewalls), or between hosts and security gateways. Since an IPSec-protected datagram is itself another IP packet, it is possible to nest security services and provide end-to-end authentication between hosts, as well as send IPSec-protected data through a tunnel (Doraswamy & Harkins, 2003).

The method of protecting IP datagrams or upper-layer protocols is by using one of the IPSec protocols, the Encapsulating Security Payload (ESP) or the Authentication

Header (AH). AH provides proof-of-data origin on received packets, data integrity, and anti-replay protection. ESP provides all that AH provides in addition to optional data confidentiality. Since ESP provides all that AH provides, one may ask, "Why use AH?" That's a good question, and is the topic of debate in the security community. The debate has shown no signs of subsiding though and AH may be relinquished, in the future. One subtle difference between the two is the scope of coverage of authentication (Frankel, 2001).

It should be noted that the ultimate security provided by AH or ESP is dependent on the cryptographic algorithms applied by them. Mandatory-to-implement algorithms are defined for conformance testing and to insure interoperability among implementations. These algorithms are generally secure, although recent advances in cryptography and the continued demonstration of Moore's law (the observation that every 18 months computing power doubles) continue to whittle away at the effective security of ciphers. The Digital Encryption Standard (DES) has depreciated for just this reason. The new Advanced Encryption Standard (AES) is taking its place.

The security services that IPSec provides requires shared keys to perform authentication and/or confidentiality. A mechanism to manually add keys for these services is mandatory to implement. This ensures interoperability of the base IPSec protocols. Of course, manual key addition scales poorly so a standard method of dynamically authenticating IPSec peers, negotiating security services, and generating shared keys is defined. This key management protocol is called IKE—the Internet Key Exchange (Kaufman, 2005).

The shared keys used with IPSec are for either a symmetric cipher (when confidentiality is needed) or for a keyed MAC (for data integrity) or for both. IPSec must be fast and existing public key technologies, such as RSA or DSS, are too slow to operate on a packet-by-packet basis. Presently, public key technology is limited to initial authentication during key exchange.

The details of the IPsec standard are given in the next chapter. The complexity of the standard is already obvious from the given information above. The detailed explanations will be kept simple as much as possible for the reader to understand the issues underneath.

CHAPTER FIVE
STANDARD BASED IP SECURITY FOR EMBEDDED SYSTEMS

To the best of our knowledge, embedded systems have no security standard covering all of them, at the IP level. There are some de-facto standards and application specific standards. Most of the standards are focused on hardware security of embedded systems and usually security issue is addressed as a part in the standard. In addition, there is no unified effort to reach a security baseline for all types of embedded systems (Stapko, 2008). On the other hand, the security of computers at IP level has always been a hot subject, with every new devastating attack announcement. Therefore, computer security standards are very advanced.

There is a suite of security standards and not just one isolated standard, concerning the securing of computers, at the IP level. In fact there is security standards at every level described, as in Chapter 4. Shortly, there exists a foundation for securing computers. Repeating all of these security standards is not possible, but SSL, TLS and IPsec are just a few to mention. The scope of the research is to fulfill security at a suitable level, to all the embedded systems. Devising a totally new approach would not gain any international support. Therefore, our work is towards forming a minimum set of security measures for embedded systems by obtaining a subset from already established and widely used standards for computers. Such a minimal security outline can increase the users' trust in embedded systems. Another gain out of this work may be the universally accepted platforms for testing and benchmarking of security, in embedded systems.

5.1 Is IPsec Standard Suitable For Embedded Systems?

While looking for a suitable reference point for security in embedded system, the IPsec standard comes out as the only full, end to end solution (Kent, 1998) (Kent & Seo, 2005). IP level security encompasses three functional areas: authentication, confidentiality, and key management. The authentication mechanism assures that a received packet was, in fact, transmitted by the party identified as the source in the packet header. In addition, this mechanism assures that the packet has not been altered in transit. The confidentiality facility enables communicating nodes to encrypt messages to prevent eavesdropping by third parties. The key management facility is concerned with the secure exchange of keys. In other words, IPsec has all that is demanded for securing embedded systems. Furthermore Figure 5.1 shows possible network set-ups where IPsec can be implemented (Stallings, 2011). The top set-up is for connecting the overall traffic of two LANs over a secure channel, via security gateways that carry out IPsec rules at either end. The middle set-up is for secure remote connection of isolated user's to a corporate LAN. It is obvious from the figure that the lowest configuration of a host to host connection is the most suitable for a controller-embedded system pair. The IPsec standard is explained below, keeping it as simple as possible. The standard is a rather complex suite of standards listed as RFC 4301-10 on Internet Engineering Task Force (IETF) web page, after the last revision to old RFCs.

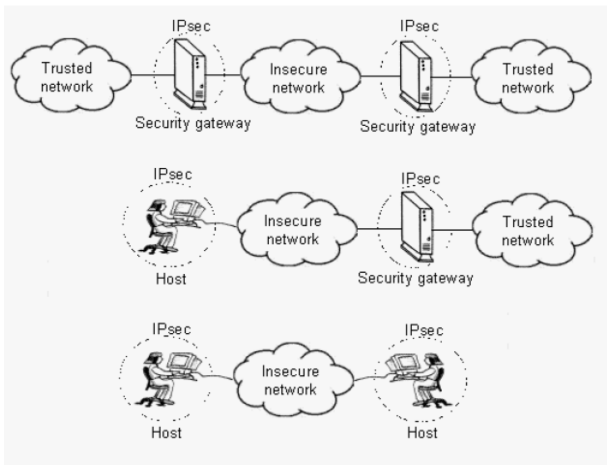


Figure 5.1 Possible IPsec Set-Ups

The services provided by IPsec (Figure 5.2) must be examined to see if it is suitable for our purposes. The table clearly shows that the expected security features such as confidentiality, integrity, authentication and key management are all supported in the Encryption Security Payload (ESP) mode (Kent, 2005). Therefore it is suitable to be used for the protection of embedded systems. The standard is further detailed in the next section, which proves better that IPsec is indeed the only choice for basing security in embedded systems.

5.2 The IPsec Standard

IPsec is designed to provide confidentiality, data integrity, access control, and data source authentication to IP datagrams. In other words; the provided solution is interoperable, high quality, cryptography-based security for IPv4 and IPv6 (Kent & Seo, 2005). The set of security services offered includes access control, connectionless integrity, data origin authentication, detection and rejection of replays (partial sequence integrity), confidentiality (via encryption) and limited traffic flow confidentiality. These services are provided at the IP layer, offering protection in a standard for all protocols that may be carried over IP (Kent & Seo, 2005). The services provided by IPsec are shown in Table 5.1. The important column is the rightmost column where both encryption and authentication is supported. This mode of IPsec is called Encryption Security Payload (ESP). It should be observed that Authentication Header (AH) mode does not provide confidentiality (Frankel, 2001). Only ESP mode is not satisfactory for embedded systems either, because data origin authentication is not supported.

Table 5.1 Services Provided By IPsec

	AH	ESP(encryption only)	ESP(encryption, plus authentication)
Access Control	✓	✓	✓
Connectionless Integrity	✓		✓
Data Origin Authentication	✓		✓
Rejection of Replayed Packets	✓	✓	✓
Confidentiality		✓	✓
Limited Traffic Flow Confidentiality		✓	✓

When IPsec is correctly implemented and deployed, it is not expected to adversely affect users, hosts, and other Internet components that do not employ IPsec for traffic protection. IPsec security protocols (AH and ESP, and IKE) are designed to be cryptographic algorithm independent (Frankel, 2001). This modularity permits selection of different sets of cryptographic algorithms as appropriate, without affecting the other parts of the implementation. For example, different user communities may select different sets of cryptographic algorithms, creating cryptographically-enforced particular groups, if required.

IPsec creates a boundary between unprotected and protected interfaces, for a host or a network. Data origin authentication and connectionless integrity are joint services, and hereafter will be referred to jointly as "integrity". This term is employed because, on a per-packet basis, the computation performed provides connectionless integrity directly; data origin authentication is provided indirectly as a result of binding the key used to verify the integrity, to the identity of the IPsec peer (Doraswamy & Harkins, 2003).

5.2.1 Bump in the Stack and Bump in the Wire Implementations

IPsec implementation in computers is carried out in two ways. These modes are shown in Figures 5.2 and 5.4. Bump in the stack (BITS) of Figure 5.2 is so called, because the operating system of a computer is patched to analyze every packet flowing through network layer, by adding an additional IPsec layer into the stack (Frankel, 2001). This is implementable in embedded systems that either have operating systems or those that are upgradable.

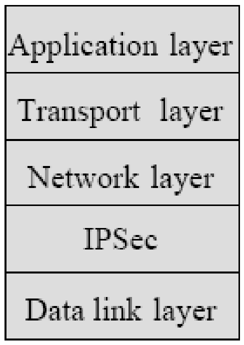


Figure 5.2 Bump in the stack (BITS) Implementation

But there are systems that cannot be intervened; or their re-configuration is not possible. These need the protection mechanism to be placed in front of them, therefore causing a bump in the wire (BITW). The layers of BITW is shown in Figure 5.4, where the network (IP) and the security (IPsec) features are interlaced in a single layer (Frankel, 2001). The physical configuration is shown in Figure 5.3, where the network wire is cut and a gateway or protector is put in between. The computers on two LANs are not directly connected. An "IPsec gateway" is inserted in between the two networks, on each side. These gateways secure the channel between the two LANs by protecting every form of traffic, by applying the IPsec standards. This is possible because the resulting IP stack has IPsec executed as

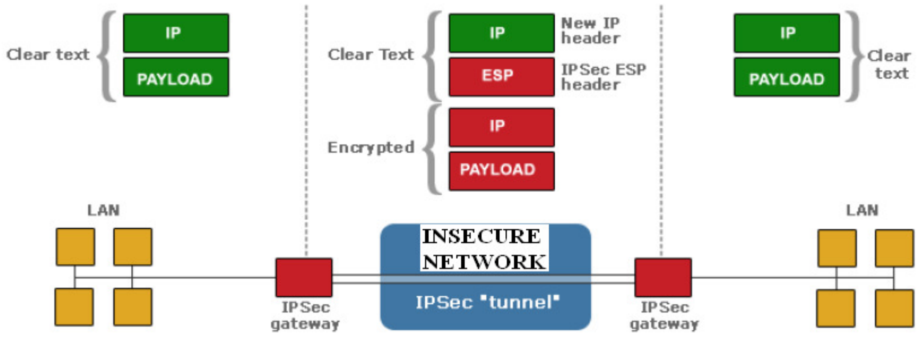


Figure 5.3 Bump in the wire implementation.

a separate layer in front. This is shown in Figure 5.4. The gateways encrypt/decrypt all of the data packets flowing from one LAN to another, thus making it impossible for an intruder to understand the communication of individual computers.

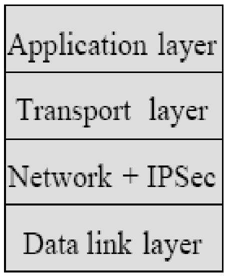


Figure 5.4 Bump in the wire implementation.

In networked embedded systems that are on a LAN the insecurity is the same but the set-up is simpler. The secure channel has to be established between the controller, which is usually a PC, and an embedded system on the same LAN. Problem arises when the embedded system which needs security is an old technology product. In this case it cannot be altered or upgraded to insert either a stack or the IPsec into its stack. There is a need for a device similar to a gateway that acts as a protector, which implements IPsec to secure the embedded system. Let us call this device a front-end protector. By nature it is a BITW implementation of IPsec. This situation is shown in Figure 5.5. The protector is placed in place of the gateway. Only one protector is needed as the controller is upgraded to contain the IPsec layer in its stack; i.e. BITS for the controller. Hence the computer and the embedded system are able to set up a secure channel via the protector.

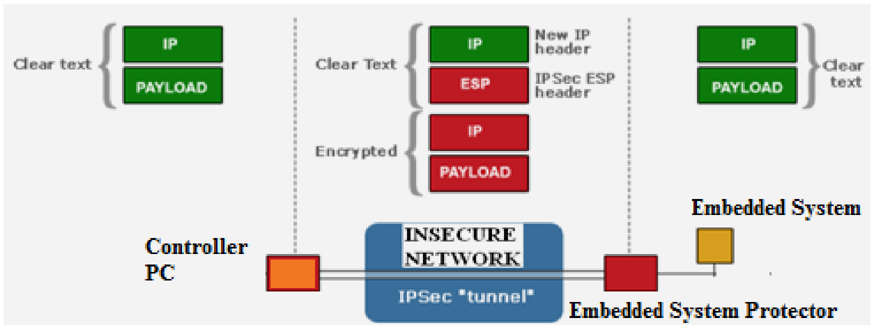


Figure 5.5 Bump in the wire implementation

5.2.2 IP Tunneling in IPsec

The confidentiality and integrity services listed in IPsec are provided by maintaining a shared state between the source and the sink of an IP datagram (Doraswamy & Harkins, 2003). This state defines, among other things, the specific services provided to the datagram, which cryptographic algorithms will be used to provide the services, and the keys used as input to the cryptographic algorithms. Once that is defined the packets are pre-pended with extra headers like ESP, as shown in Figure 5.5. The services provided by IPsec are given in Table 5.2.

Table 5.2 AH and ESP Services Provided.

	Transport Mode SA	**Tunnel Mode SA**
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts inner IP packet. Authenticates inner IP packet.

5.2.3 Internet Key Exchange Version 2 of IPsec

Internet Key Exchange version two (IKEv2) combines three, old separate documents of IPsec (Kaufman, 2005), which were highly criticized previously. The standard defines the rules that should be followed between two peers for obtaining a set of session keys. The exchange consists of 6 steps or messages in total, shown in Figure 5.6. Messages are defined in pairs, as challenge (suffix _C) and response (suffix _R).

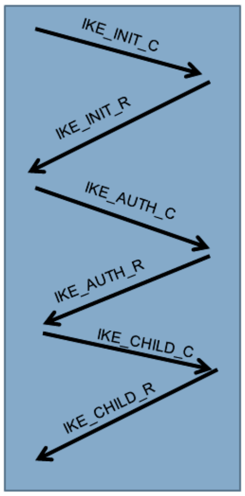


Figure 5.6 IKEv2 Key Exchange Protocol of IPsec

Any party can start the exchange by sending a challenge or request to a party it wants to communicate. In other words, the parties are assumed to be total strangers to each other at the beginning of the dialogue. Or the opposite is assumed to be a possible masquerader, therefore the identity is not revealed until the third message. Every challenge is ruled to have a response and there are strict rules when to answer or not to answer some challenges.

Neither the previous documents; nor NAT Traversal, Legacy Authentication and Remote Add Acquision will be commented on, as they are not the related to this research. They are not related because the controller and the embedded system reside on the same local area network, and there is no need for translating subnet address the way LAN addresses are translated. Neither the Legacy Authentication, nor the Remote Add Acquision are used because the parties are not remote from each other. The rest of the details of the IKEv2 will be given in the next section, where its modification for embedded systems is discussed.

5.2.4 Applying the Rules of IPsec IKEv2 to Embedded Systems

IPsec's IKEv2 is reported to be appropriate for use in particular situations, by the RFC itself (Kaufman, 2005). The range of choices defines precisely how IPsec is to be implemented. Therefore; any work that relies on IPsec's IKEv2 is expected to specify the following:

(a) What selectors; i.e. what addresses, port numbers etc., are to be used by the initiator of the conversation (the client, in client-server architectures)?

(b) What IPsec protocol is to be used: AH or ESP? What mode is to be employed: transport mode or tunnel mode?

(c) What form of key management is appropriate?

(d) What security policy database entry types should be used by the responder; i.e. the server, when deciding whether or not to accept the IPsec connection request?

(e) What form of identification and authentication should be used?

(f) Which of the many variants of IKE must be supported?

(g) Is suitable IPsec support available in configurations of the products that would employ IPsec?

The requirements above are answered here. First of all the addresses and port numbers of the initiator are known, because it is either the controller or the embedded system with a pre-defined address and port of application. It has been shown that AH mode is inherent in the ESP tunneling mode. Therefore AH is dropped in our work. Transport is not secure enough for the embedded systems. Therefore tunneling that uses ESP mode is the choice. Key management is based on the pre-shared secret model outlined in the RFC itself. The keys are formed as drafted in the RFC. Therefore key management is also within standards. IP address and ID number will be used to apply the rules of the RFC. AES is chosen to be used in the authenticator production (Frankel & Herbert, 2003). Only pre-shared secret variant of IKE will be used because any other option is not secure and too resource demanding for embedded systems. And "Yes", suitable support is available in the form of hardware AES engines that can be easily fitted into embedded systems, which off-loads the microcontroller and speeds up the encryption processing (Feldhofer & Wolkerstorfer, 2009).

5.3 Simplifying IPsec to Fit In Embedded Systems

The Security Considerations sections of RFC 4301 strongly recommend the use of IPsec, unaltered (Kent & Seo, 2005). While this is sometimes correct, often it will leave users without real, interoperable security mechanisms. IPsec is often unavailable in the likely endpoints. Even if it is available, it may not provide the proper granularity of protection. Finally, even if IPsec is available and appropriate in a system, the document mandating IPsec needs to specify just how it is to be used; which it doesn't. The design of security protocols for a large set of devices is a subtle and difficult art. Therefore, it is often advantageous to break down the set into subsets and apply a solution more effectively tailored for each set.

The cautions about specifying the use of IPsec should not be taken to mean to invent a new security protocol, for each new application. If IPsec is a bad choice, using another standardized, well-understood security protocol will almost always give the best results for both implementation and deployment. But if IPsec is a good starting point, it can be tailored to address the security expected. Security protocols are very hard to design and rolling out a new one requires extensive theoretical and practical work to verify its security properties. The design process will incur both delay and also uncertainty among the community. It appears as if it would be best to build an approach based on an internationally accepted standard.

With the above recommendations in mind, the first work consisted of examining of the old and new IPsec RFCs. Literature survey showed that the RFCs 2407, 2408, 2409 were replaced by a single RFC 4306 (Kaufman, 2005). The three old RFCs were joined in one document where the Domain of Interpretation (DOI) of IPsec was reiterated. These RFCs have been studied in detail and 87 points have been noted down. Thus, the first draft of a DOI for embedded systems (DOIES) has been prepared. The third effort has been the analysis of the IPsec key exchange and its authentication steps. At the end of the analysis a simplified IKEv2 exchange has been devised. The next two sections describe how the described simplifications have been made, respectively.

5.3.1 Simplification on the Organization of IPsec

The organization of IPsec shown in Figure 5.7 is kept structurally unaltered, but reduced in terms of its elements redundant for embedded systems. In the first step; the database keeping the records of established associations -Secure Association Database (SAD) - and the database keeping a record of established policies -Security Policies Database (SPD) - are eliminated; because the embedded system is not going to be in communication with many controllers but only one. Secondly, only one security association will be in use during the communication. The controller might need to keep these constructs if it is going to be connected to many embedded systems. But this again is not likely because the Security Associations (SAs) will only differ in the keys negotiated. Keeping a database for a limited number of keys is not efficient.

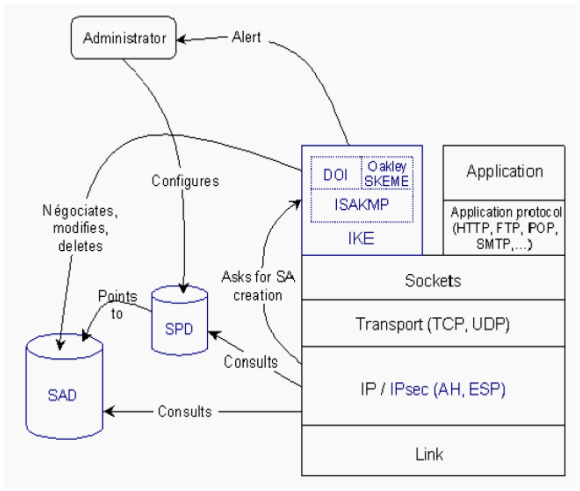


Figure 5.7 Organization of IPsec's IKEv2

First phase of simplification results in the structure shown in Figure 5.8. Here reduced versions of SPD and SAD are shown inside the operating system, meaning other software means can be used to keep the security association information. Still, further simplification can be achieved, as shown in Figure 5.9.

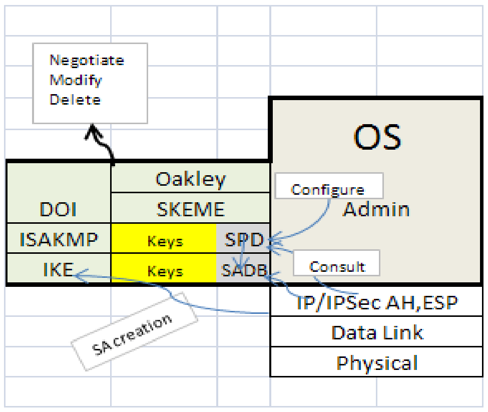
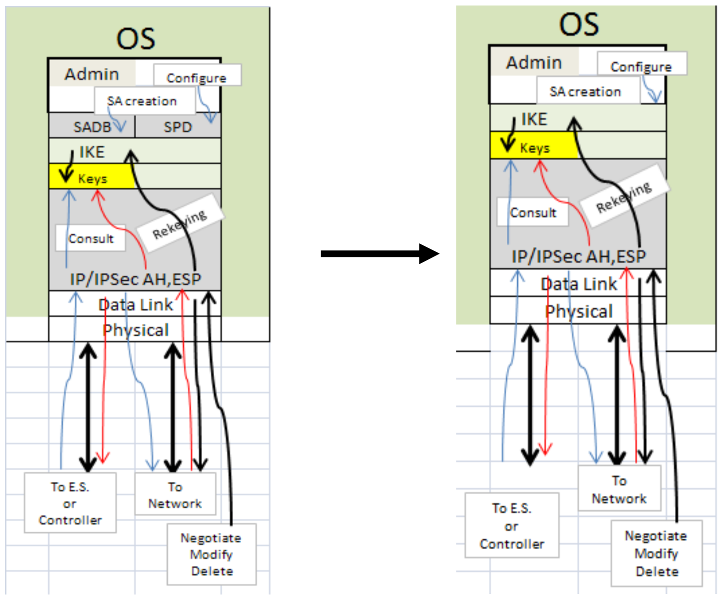


Figure 5.8 Bump in the wire implementation

The final reduction is the removal of all database programs from the embedded systems. This can be achieved by keeping only the keys in a secure file and making the rest of the SA attributes constant for a secure channel. This simplification is shown as a transition from Figure 5.9a to Figure 5.9b. Observe the enormous simplification from the original organization of IKEv2 of Figure 5.7.



(a) (b)

Figure 5.9 Further Simplified IPsec Organization for Embedded Systems.

The reduction of SAD and SPD has offered a further simplification, as the need for a database management interface has disappeared. This is a big simplification, in terms of development time, memory and cost.

5.3.2 Simplification of IKE Exchanges and The Payload Packets

The omission of database constructs from the IPsec greatly reduces the demand for secondary memory space and search processing through the database. IKEv2's RFC 4306 describes the rules that govern the exchange of parameters, the methods of key generation and the way the parameters are turned into payload packets, in IKEv2. The IKEv2 exchange of IPsec is shown in Figure 5.6 (Kaufman, 2005). The exchange takes place under strict rules and names are given to each message. The initial exchanges of the initiator and the responder are named as IKE_INIT. Initiator's challenge is denoted by a _C suffix and the responder's response by a _R. Every challenge has a response in IKEv2. The communicating partners can be the Initiator or the Responder, or vice versa in IPsec. But in our case it can be agreed that either the controller or the embedded system can be always the initiator. The controller is chosen to be the initiator in our research because a lot of initialization load is taken away from the embedded system, by doing so. The embedded system just turns on and waits for a call. This greatly off loads the embedded system.

The contents of the first of the six message exchanges shown in Figure 5.6 are given in detail in Figure 5.10. This is expected to be a challenge from the initiator (controller of the embedded system) to the responder (the embedded system). The parameters are the message header (HDR), the proposed attributes for the security association to be established (SA_1), the public Diffie-Hellman (Diffie & Hellman, 1976) key of the initiator (KE_i), and the nonce of the initiator (N_i).

The corresponding message packet formed related to the contents is shown in the diagram just below the challenge. The first part is the header of the IPsec's IKE. It chains to the proposed attributes. Close study reveals that the more the proposed attributes the longer the chain and thus the longer is the message. This means increased complexity, processing and message length. The last attribute proposal is chained to the public key payload, which in turn is chained to the nonce payload. By substituting the numerical values for each field given in RFC 4306, the payloads are constructed. The message transform into the form shown on the right hand side; which is in fact a packet of numbers.

INITIATOR **RESPONDER**
HDR, SAi$_1$, KE$_i$, N$_i$ →→→

IKE_SA Initiator SPI				
IKE_SA Responder SPI				
Next Payload=SAi1	MjVer	MnVer	Exh. Ty=IKE_SA_INIT	Flags
Message ID				
Length				
Next Payload=KEi1	C! Reserved	Payload Length		
last proposal= 0	Reserved	Proposal Length		
Proposal #1	Protocol ID=IKE	SPI Size=16	# of Trans=4	
SPI (variableX fixed)				
Last or 3	Reserved	Transform Length		
Transform Type=1	Reserved	Transform ID=AES=12		
Last or 3	Reserved	Transform Length		
Transform Type=2	Reserved	Transform ID= PRF_HMAC_SHA1=2		
Last or 3	Reserved	Transform Length		
Transform Type=3	Reserved	Transform ID=HMAC_SHA1_96=2		
Last or 3	Reserved	Transform Length		
Transform Type=4	Reserved	Transform ID=DH GROUP= 1		
Next Payload=Ni	C! Reserved	Payload Length		
DH Group =1	Reserved			
Key Exch. Data = g^x				
0	C! Reserved	Payload Length		
Ni Payload Data = nonce i				

(a) Regular Chain of Payloads

INITIATOR **RESPONDER**
HDR, SAi$_1$, KE$_i$, N$_i$ →→→

OOO1				
OOO2				
33	**2**	**2**	**34**	Flags
Message ID				
Length				
34	C! Reserved	Payload Length		
0	Reserved	Proposal Length		
1	1	**16**	**4**	
SPI (variableX fixed)				
3	Reserved	Transform Length		
1	Reserved	**12**		
3	Reserved	Transform Length		
2	Reserved	**2**		
3	Reserved	Transform Length		
3	Reserved	**2**		
0	Reserved	Transform Length		
4	Reserved	**1**		
40	C! Reserved	Payload Length		
1	Reserved			
g^x				
0	C! Reserved	Payload Length		
nonce i				

(b) Simplified

Figure 5.10 Simplifications of IPsec's IKEv2 Exchange Payloads.

The attention of the reader is drawn to the fact that, the length of the whole message is variable, depending on the number of proposals made. Needless to say, the construction of the challenge message is complicated and computationally difficult. The computations increase drastically when messages are encrypted and then sent, in the next steps.

A new DOIES can be prepared where the attribute negotiation, SPI values, message IDs are made constant and some rules can be simplified; to yield a simpler payload chain. The message contents of the key exchange; namely, the header, payload fields can be simplified and some procedures can be rendered redundant. A lot of payloads are not needed for embedded system communication, because of the nature of the one-to-one communication with the controller. Some payload types are not necessary in embedded systems and some fields of the payloads are unnecessary, when only one security suite is used for one channel. As an example consider AES for as the only option for encryption.

IPsec's IKEv2 performs mutual authentication between two parties and establishes a security association (SA) that contains the shared secret information used to efficiently establish the Encapsulating Security Payload (ESP) or the Authentication Header (AH) and a set of cryptographic algorithms to be used by the SAs to protect the traffic that they carry. An initiator proposes one or more selectors by listing the algorithms it supports which can be combined in a mix-and-match fashion. This complicates the agreement on key generation and choice of algorithms. There is no need for agreement between two partners aware of each other, as in the case of an embedded system and its controller. Therefore, the whole negotiation of algorithm choices and other selectors can be dropped.

For the purpose of simplifying both IPsec and IKEv2, the RFC 4301 and 4306 are examined and simplified step by step. It should be stated that none of the rules have been eliminated. All authenticator calculations, key generation and payload construction rules have not been altered. Below both the simplifications made and the parts left unaltered referring to the RFCs are argued in long list, point by point:

1. One of the major simplifications is the removal of AH from embedded systems' security, as it does not offer encryption. AH mode itself is hardly used at all in computers either (Wouters & Bantoft, 2006).

2. SAD and search through a database is omitted because association entries do not require the use of a database.

3. There is no ICMP support in our research because the embedded system is

not expected to reply ICMP requests as we are trying to hide it.

4. Alternative configuration set-ups of IPsec is reduced to only host to host configuration and rules, in our work.

5. IPsec does not support the manual establishment of the first shared state as it does not scale well, as many computers that are totally strangers to each other. But this is not the case for a limited number of embedded systems in a single establishment. Our target configuration is a one-to-one and rarely one-to-many communication configuration. There is no many-to-many communication. Therefore, the manual agreement of the initial shared state does not pose scalability problems for our purposes. Hence, the manual set of relationships fits well because after the first manual setup later rekeying state is adequate for this configuration.

6. The manual state setting above also leads to the elimination of a SPD database as well. There is usually only one tunnel between an embedded system and its controller (rarely more than one), thus all traffic is carried via a single SA and a single set of security services. This eliminates the need for keeping an SPD.

7. The IPsec standard supports a large number of alternative selector parameters to provide granularity for SA. But there is no need to support any different selector parameters in embedded system implementations. One single SA based on a strong selector like AES can serve all encryption, pseudo-random function and integrity function demands.

8. IPsec requires support for both manual and automated distribution of keys. There is no obligation of automatic key distribution, thus the master-slave configuration of controller PCs and embedded systems does not violate the key distribution rules of IKEv2.

9. All IKE communications consist of pairs of messages: a request and a response. The pair is called an "exchange". The first messages establishing an SA are called IKE_SA_INIT and IKE_AUTH exchanges. Subsequent IKE exchanges are called CREATE_CHILD_SA or INFORMATIONAL exchanges. Our research keeps these standard names intact. In the common case, there is a single IKE_SA_INIT exchange and a single IKE_AUTH exchange (a total of four messages) to establish the IKE_SA and the first CHILD_SA. Our work abides by these rules. But the CHILD_SAs are simplified to rekeying in our implementation, as no other new parameter negotiation is necessary.

10. The subsequent exchanges cannot be used until the initial exchanges of an IKEv2 have completed. There is no need to change these basic rules and they are preserved in our work.

11. Among the possible configuration scenarios of IPsec, our implementation is fixed to the host-to-host configuration. The other scenarios are not needed in our controller PC-controlled embedded system set up.

12. The second pair of messages (IKE_AUTH) authenticates the previous messages, exchange identities and certificates, and establish the first CHILD_SA. Parts of these messages are encrypted and integrity protected with keys established through the IKE_SA_INIT exchange, so the identities are hidden from eavesdroppers and all fields in all the messages are authenticated. These rules are preserved as will be shown in the following section where the fields of the messages also simplify, after these decisions.

13. The payloads contained in the messages are preserved. Their notation, types and chaining orders are preserved but after the decisions made above most payloads are not needed, like the certificate related payloads are dropped completely.

14. The initial exchange between the Initiator and the responder is as follows:

Initiator		**Responder**	
HDR, SA_{i1}, KE_i, N_i	==>		(1)

The SAi1 payload states the cryptographic algorithms the initiator supports for the IKE_SA, which is dropped due to the above simplifications described. The KE payload sends the initiator's Diffie-Hellman value and Ni is the initiator's nonce, which are kept. Work shows keeping SA_1 is just a repetition overhead, as both the controller and the embedded system support the same selectors only. The simplified message is now:

Initiator		**Responder**	
HDR', KE_i, N_i	==>		(2)

HDR contains the Security Parameter Indexes (SPIs), version numbers, and flags of various sorts. These can be simplified because there is no need for an SPI as there is only one SA. But removing the SPI can be a violation of the message field integrity. For this reason only, the SPI can be kept in place. The new header is shown as HDR' in (2).

15. The IKEv2 response is as follows:

Initiator		**Responder**	
	<==	HDR, SA_{r1}, KE_r, N_r, [CERTREQ]	(3)

Normally, the responder chooses a cryptographic suite from the initiator's offered choices and expresses that choice in the SA_{r1} payload, completes the Diffie-Hellman exchange with the KE_r payload, and sends its nonce in the N_r payload. Immediately, SA_{r1} is redundant and dropped as SA_{i1} is no longer used. Dropping SA_{r1}, results in the elimination of response decisions and redundant negotiations. The new response simplifies to:

Initiator		**Responder**	
	<==	HDR', KE_r, N_r	(4)

16. At this point in the negotiation, each party can generate SKEYSEED, from which all keys are derived for that SA. All but the headers of all the messages that follow are encrypted and integrity protected. The keys used for the encryption and integrity protection are derived from SKEYSEED and are known as SK_e (encryption) and SK_a (authentication, i.e. integrity protection). A separate SK_e and SK_a is computed for each direction. In addition to the keys SK_e and SK_a derived from the DH value for protection of the IKE_SA, another quantity SK_d is derived and used for derivation of further keying material for CHILD_SAs. The notation SK { ... } indicates that these payloads are encrypted and integrity protected using that direction's SK_e and SK_a. These notations and rules must be preserved and are preserved, otherwise standard compatibility is lost.

17. There is a lot of simplification in the authentication phase. The challenge of this phase is :

Initiator	**Responder**	
HDR, SK {ID_i, [CERT,] [CERTREQ,] [ID_r,] AUTH, SA_{i2}, TS_i, TS_r} ==>		(5)

The initiator asserts its identity with the ID_i payload, proves knowledge of the secret corresponding to ID_i and integrity protects the contents of the first message using the AUTH payload. The optional payload ID_r enables the initiator to specify which of the responder's identities it wants to talk to. This is useful when the machine on which the responder is running is hosting multiple identities at the same IP address. As only pre-shared secrets will be used all public certificates are dropped. There is no need for SA_{i2}, TSi or TS_r negotiation for the CHILD_SA, as these have been already fixed. The new message is reduced to:

Initiator **Responder**

HDR, SK{ID_i, ID_r, AUTH, } ==> (6)

Apart from the reduced number of parameters in the message, the amount of fields to be encrypted has also reduced. This means a considerable amount of reduction is computation.

18. Similarly the response is also simplified with similar arguments, from :

Initiator **Responder**

<== HDR, SK {ID_r, [CERT,] AUTH,SA_{r2}, TS_i, TS_r} (7)

To the reduced form:

Initiator **Responder**

<== HDR, SK {ID_r, AUTH} (8)

The responder completes the exchange, by asserting its identity with the ID_r payload, authenticating its identity and protecting the integrity of the message with the AUTH payload.

19. The recipients of simplified messages 6 and 8 verify that all signatures and MACs are computed correctly to prove the keys used to generate the AUTH payload. This check is standard. The only thing left to mention here is because of the eliminated fields the payloads will be dramatically simplified.

20. The CREATE_CHILD_SA Exchange consists of another request/response pair. It may be initiated by either end but in our implementation this is simplified to the secure choice of always controller initiation. All messages following the initial exchange are cryptographically protected by the

cryptographic algorithm and keys negotiated in the first messages of the IKE. These subsequent messages use the Encrypted Payload syntax. All subsequent messages include an Encrypted Payload, even if they are "empty". This standard is preserved.

21. The original CREATE_CHILD_SA request may optionally contain a KE payload for an additional Diffie-Hellman exchange to enable stronger guarantees of forward secrecy. As one of our primary concerns is forward secrecy, the requests always contain a KE payload, in our design. The keying material for the CHILD_SA is a function of SK_d key established before. Either the old nonces are used or new ones are exchanged, normally in IPsec. For stronger security new nonces are used in our implementation. The CREATE_CHILD_SA request is shown (9), which is changed into (10), of only a three parameter version:

Initiator **Responder**

HDR, SK {[N], SA, N_i, [KE_i],[TS_i, TS_r]} ==> (9)

Initiator **Responder**

HDR, SK {N_i, KE_i } ==> (10)

22. In IPsec; if CREATE_CHILD_SA exchange is rekeying an existing SA the leading N payload of type REKEY_SA must identify the SA being rekeyed. This is not necessary in ours because there is only one SA. Rekeying is always an exchange of new nonces and public keys to generate a new session key. This greatly simplifies rekeying complexities.

23. In IPsec, the SA offers include different Diffie-Hellman groups (Stallings, 2011), which results in many complexities. In our implementation the controller-embedded system pair has only one fixed DH group, which requires no negotiation. This also greatly reduces the payload fields and exchanges.

24. The message following the header is encrypted and the message including the header is integrity protected using the cryptographic algorithms negotiated for the IKE_SA, this is a property that cannot be simplified. Only in our work the cryptographic algorithm is AES and needs no negotiation as it is pre-decided from the start.

25. The CREATE_CHILD_SA response of IPsec is complex as it is dependent on the challenge of the responder. If the negotiation does not end up in agreement in the first challenge-response exchange, then exchange continues. Our simplification overcomes this negotiation over-load as shown below:

Initiator **Responder**

$< ==$ HDR, SK {SA, N_r, [KE_r],[TS_i, TS_r]} (11)

Initiator **Responder**

$< ==$ HDR, SK {N_r, KE_r} (12)

26. The traffic selectors for the traffic of the SA are specified in the TS_i, TS_r payloads. This is not an issue in our case and the traffic selectors are omitted.

27. The INFORMATIONAL exchanges occur only after the initial exchanges and are cryptographically protected with the negotiated keys. The request message in an INFORMATIONAL exchange may contain no payloads. This is the expected way an endpoint can ask the other endpoint to verify that it is alive. These rules are obeyed in our work but only the controller questions the embedded system if it is alive. This strategy also results in simplifications.

28. In IPsec for computers, ESP and AH SAs always exist in pairs. There is one SA in each direction. When a SA is closed, both members of the pair MUST be closed. This detail does not cause problems in our work, because the single SAs are identical for each direction. There is no need to keep the same information; hence a single SA is enough.

29. Deleting an SA in IPsec requires an informational exchange with one or more delete payloads is sent listing the SPIs (as they would be expected in the headers of inbound packets) of the SAs to be deleted. This complex activity is greatly simplified in our work. There is only one SA to be deleted. Hence, whatever the reason the deletion of an SA is very easy in our implementation.

30. If an encrypted IKE packet arrives on port 500 or 4500 with an unrecognized SPI, it could be because the receiving node has recently crashed and lost state or because of some other system malfunction or attack. If the receiving node has an active IKE_SA to the IP address from whence the packet came, it may send a notification of the wayward packet over that IKE_SA in an informational exchange. If it does not have such an IKE_SA, it may send an Informational message without cryptographic protection to the source IP address. Such a message is not part of an informational exchange, and the receiving node does not respond to it. Doing so could cause a message loop. The above events are true for IPsec and have to be naturally obeyed. But, in our implementation if either of the parties reset then the IKEv2 secure channel is reconstructed from the very beginning. Although this may sound as a load on an embedded system that has not reset, nevertheless it removes any hesitation of an attack. This also helps avoiding any unnecessary informational exchange.

31. IP fragmentation, NAT and/or firewall implementations are not related to our work since the controller and the embedded system reside on the same LAN.

32. IKEv2 implementations are expected to be able to send, receive, and process IKE messages that are up to 1280 bytes long and to send, receive, and process messages that are up to 3000 bytes long. This is not a critical burden to support therefore it is preserved in our work.

33. For every pair of IKE messages, the initiator is responsible for retransmission in the event of a timeout. The responder never retransmits a response unless it receives a retransmission of the request. To resist attacks this rule is preserved.

The only simplification in our work is the controller is always the initiator.

34. IKE is a reliable protocol, as the initiator must retransmit a request until either it receives a corresponding reply or it deems the IKE security association has failed. This is a critical rule and must be obeyed at all times.

35. Each endpoint in the IKE Security Association maintains two "current" Message IDs: the next one to be used for a request it initiates and the next one it expects to see in a request from the other end. Responses always contain the same message ID as the corresponding request. This rule is followed in our implementation with the simplification that only the controller keeps two Message IDs. The embedded system keeps only one.

36. In regular IPsec, an IKEv2 endpoint is prepared to accept and process multiple requests while it has a request outstanding. In our simplification only the controller processes requests, the embedded system does not.

37. Rekeying an IKE_SA resets the sequence numbers. This is strictly obeyed and does not cause any overhead.

38. An IKE endpoint is allowed to forget all of its state associated with an IKE_SA and the collection of corresponding CHILD_SAs at any time. This is the anticipated behavior in the event of an endpoint crash and restart, in our work as well. In our work there is only one SA to forget or reconstruct.

39. To find out if the other is alive send an acknowledgement or empty message. It is suggested that messages be retransmitted at least a dozen times over a period of at least several minutes before giving up on an SA. But this is an extremely long period in a LAN environment. As IPsec anticipates different environments require different rules, our work is allowed to be stricter on retransmission. It would be safer if the controller only retransmits requests and retransmission stops after ten times.

40. There is a Denial of Service attack on the initiator of an IKE_SA that can be avoided if the initiator takes the proper care. Since the first two messages of an SA setup are not cryptographically protected, an attacker could respond to the initiator's message before the genuine responder and poison the connection setup attempt. To prevent this, the initiator may accept multiple responses to its first message, treat each as potentially legitimate, and respond to it. Then after a pre-decided number of times the controller has the capability to resist a DoS attack. As the embedded system is not the initiator in our work, this problem is avoided before it starts.

41. In IPsec here is no reason to negotiate and agree upon an SA lifetime. But a lifetime is viable and can be a pre-agreed number of times of using the same key; after which a new key is generated through a CHILD_SA exchange.

42. An IKE endpoint can delete any inactive CHILD_SA in IPsec. It then sends delete payload notification to the other end point. In our case this needs consideration. An embedded system cannot single handedly decide to delete any SA; unless it resets. Only the controller can. Therefore, if the controller decides to, it will inform the embedded system and initiate a rekeying or a total restart of the secure tunnel. This offloads the embedded system.

43. In IPsec an active attacker can trick two IKEv2-capable nodes into speaking v1; which has shown some weaknesses. In our implementation there is no negotiation of version number. Therefore such a weakness does not exist in our work. In addition, there is simplification due to fixed version usage.

44. An expected attack against IKE is state and CPU exhaustion. The target of the attack is flooded with initiation requests from forged IP addresses. This attack can be made less effective if the responder uses minimal CPU and commits no state to an SA; until it knows the initiator can receive packets at the address from which it claims to be sending them. The responder is the embedded system in our work and it is expecting requests from the known IP

address of the Controller. Any request from other IP numbers is not replied. If the controller IP is forged, a small counter can catch the number of repetitions and stop responding. To raise an alarm the embedded system replies using an another pre-agreed channel. Finally a re-initialization of the secure channel is necessary. Cookie generation is not followed in our implementation.

45. The payload type known as "SA" indicates a proposal for a set of choices of IPsec protocols (IKE, ESP, and/or AH) as well as cryptographic algorithms associated with each protocol. Thus an SA payload consists of one or more proposals. Our work restricts the choice to only one. Therefore, the simplest reduction is to drop SA payload.

46. Re-establishment of security associations to take the place of ones that expire is referred to as "rekeying". The ability to rekey SAs without restarting the entire IKE_SA is optional. In or work we propose to re-key after a key is used for a pre-set number of times. SAs are rekeyed proactively, i.e., the new SA is established before the old one expires and becomes unusable.

47. Traffic Selector (TS) payloads allow endpoints to communicate some of the information from their SPD to their peers. Traffic selectors are not used in our implementation

48. The IPsec IKEv2 dictates the following about the nonces. The IKE_SA_INIT messages each contain a nonce. These nonces are used as inputs to cryptographic functions. The CREATE_CHILD_SA request and the CREATE_CHILD_SA response also contain nonces. These nonces are used to add freshness to the key derivation technique used to obtain keys for CHILD_SA, and to ensure creation of strong pseudorandom bits from the Diffie-Hellman key. Nonces used in IKEv2 are randomly chosen, at least 128 bits in size and at least half the key size of the negotiated prf. ("prf" refers to "pseudo-random function", one of the cryptographic algorithms negotiated in the IKE exchange.). As the decision is using AES in our implementation for all

algorithms, all of the rules are obeyed. The AES inputs, outputs and the key are all 128 bits. The nonces can be 64 bit but they are expanded to 128 bits based on a pre-decided algorithm.

49. IKE runs over UDP ports 500 and 4500, and implicitly sets up ESP associations for the same IP addresses it runs over. The IP addresses and ports in the outer header are not cryptographically protected. The same is true for our work as well. But for embedded systems where a protector is used as a security front end, the IP of the embedded system is protected.

50. IKE generates keying material using an ephemeral Diffie-Hellman exchange in order to gain the property of "perfect forward secrecy". This means that once a connection is closed and its corresponding keys are forgotten, even someone who has recorded all of the data from the connection and gets access to all of the long-term keys of the two endpoints cannot reconstruct the keys used to protect the conversation without doing a brute force search of the session key space. Achieving perfect forward secrecy requires that when a connection is closed, each endpoint is expected to forget not only the keys used by the connection but also any information that could be used to recompute those keys. In particular, it has to forget the secrets used in the Diffie-Hellman calculation and any state that may persist in the state of a pseudo-random number generator that could be used to re-compute the Diffie-Hellman secrets. The above is the exact way our implementation works. However, since the computing of Diffie-Hellman exponentials is computationally expensive, IPsec allows the reuse those exponentials for multiple connection setups. This opens the avenues for attacks and is not allowed in our work.

51. In IPsec the keying material is always derived as the output of the negotiated prf algorithm. Since the amount of keying material needed is greater than the size of the output of the prf algorithm, the prf is used iteratively. Our work uses repeated AES encryption to get the keys, fully in compliant with the standard.

52. The peers are authenticated by having each sign (or MAC using a shared secret as the key) a block of data. It is critical to the security of the exchange that each side sign the other side's nonce. This is strictly obeyed.

53. The calculation of the authenticator AUTH = prf (prf (Shared Secret, "Key Pad for IKEv2"), <msg octets>) is also fully obeyed. Any simplification may break the security of the whole exchange.

54. The CREATE_CHILD_SA exchange can be used to rekey an existing IKE_SA. New initiator and responder SPIs are supplied in the SPI fields. SKEYSEED for the new IKE_SA is computed using SK_d from the existing IKE_SA as follows: SKEYSEED = prf (SK_d (old), [g^ir (new)] | N_i | N_r) These are fundamental rules and cannot be changed for the sake of simplification. They are kept intact in our work.

55. Requesting an Internal Address on a Remote Network is not related with our implementation, therefore dropped.

56. Error Handling rules and payloads are reduced to resetting the embedded system and restarting a new IKEv2 exchange from the scratch. This greatly simplifies the chores of the embedded system.

57. If a node receives a message on UDP port 500 or 4500 outside the context of an IKE_SA known to it (and not a request to start one), it may be the result of a recent crash of the node. If the message is marked as a response, the node audits the suspicious event but does not respond. If the message is marked as a request, the node audits the suspicious event and sends a response. These rules are obeyed; it is the controller who controls the dialogue. The embedded system counts the number of responds repeated and raises an alarm if suspicious events multiply to a pre-determined number.

58. IPcomp is not used in our implementation. This is because there is no speed

bottleneck on a LAN subnet. Secondly, the embedded systems do not exchange a lot of data as in the case of a computer. Only control data are sent to an embedded system. Usually there is no long file transfer processes.

59. NAT Traversal or ECN are not related to our work on a LAN and they are dropped for simplification.

60. The UDP ports used for IKEv2 exchange are preserved, as well as the header and payload formats of the exchanges and the 4 octets of leading zeros on port 4500.

61. An encrypted payload is the last payload in IPsec. There are no multi-peers or multi-SAs in our implementation. Therefore, this rule does not cause any overhead in our implementation.

62. The major and minor versions are constant, in the header of our implementation. They can be either dropped or disregarded while analyzing the payloads.

63. SPIs can be simplified to 32 leading zeros and the remaining 32 bits can be dynamic.

64. Chaining of payloads is left untouched, as it is essence of IKE.

65. Exchange types and payload types are standard and cannot be changed.

66. The flags in the headers are kept except the reserved bits of IPsec can be used for purposes of DOIES implementations.

67. The generic payload header is left untouched except the Critical Bit. In our implementation its value is fixed to “0” always.

68. In IPsec Security Association Payload (SA payload) may contain multiple proposals. Each proposal may contain multiple IPsec protocols (where a protocol is IKE, ESP, or AH). Each protocol may contain multiple transforms, and each transform may contain multiple attributes. No SA is needed in our implementation partners know what they agree on. Therefore SA is dropped completely. Hence the lengths of the messages are pre-determined in our implementation. The reason for the complexity and the hierarchy is to allow for multiple possible combinations of algorithms to be encoded in a single IPsec SA, which is unnecessary in our work.

69. IKEv2 generally has four transforms: a Diffie-Hellman group, an integrity check algorithm, a prf algorithm, and an encryption algorithm. Negotiation takes place via exchanging SAs. This is dramatically simplified in our work. All these are pre-decided, therefore use of either a pre-determined Sa exchange takes place or the SA is dropped. Hence the users have the chance to buy an AES secured embedded system. Fixed but secure system until AES is broken.

70. The Transform Types in the SA payload proposing the establishment of an SA are preserved. But since the SAs are dropped in our work the transform types 1, 2, 3 4, 5 are pre-determined. The parties are presupposed to have noted these down and the IKEv2 exchange starts as these have been pre-agreed upon.

71. All implementations of IKEv2 are expected to include a management facility that allows specification (by a user or system administrator) of Diffie-Hellman (DH) parameters (the generator, modulus, and exponent lengths and values) for new DH groups. Implementations provide a management interface via which these parameters and the associated transform IDs may be entered (by a user or system administrator), to enable negotiating such groups. The management facility is expected to enable a user or system administrator to specify the suites that are acceptable for use with IKE. There is no need left for a management facility in our implementation, as the above management of choices has been simplified. Even if a management facility exists it is very

short and additionally may only reside on the controller PC. Hence the management facility load of the embedded system is non-existent or nearly zero; if a very primitive management facility menu is used.

72. The key length is a big issue in IPsec. By adopting AES-128, our work has fallen within the standards of all key length and encryption related RFCs.

73. Attribute Negotiation of IPsec has also been dropped like the Transform Types. This is another simplification reached.

74. The key exchange and Identification payloads are preserved. The key exchange payload is constructed by copying one's Diffie-Hellman public value into the "Key Exchange Data" portion of the payload. The ID of the partners only uses IPv4 in our work, as we are on a local subnet. The Certificate payload is not use in pour local area embedded systems.

75. The Authentication Payload, denoted AUTH in this memo, contains data used for authentication purposes. The syntax of the Authentication data varies according to the Auth Method used. Auth Method (1 octet) field specifies the method of authentication used. Only the shared key method is used in our implementation, which makes this field a constant.

76. The size of a Nonce is expected to be between 16 and 256 octets inclusive and the Nonce values are not repeated or reused. The embedded systems can generate a 62 bit nonce and extend it to 128 bits through a pre-agreed method. The size of the Nonces is therefore fixed 16 octets in our implementation.

77. Among the Notify Payload Types only a subset is meaningful for our work after the simplifications. This reduced number of notification payloads results in further simplification of the overall DOI for the embedded systems.

78. The Delete Payload contains a protocol specific security association identifier

of a no longer valid SA that the sender has removed from its security association database. It is possible to send multiple SPIs in a Delete payload, in IPsec. Since there is only one SA and its related SPI, in our implementation, a pre-agreed, simple, reduced delete payload is enough.

79. Vendor ID or Traffic Selector Payloads have been omitted in our work.

80. The Encrypted Payload, denoted as SK{...} or E (), contains other payloads in encrypted form. The Encrypted Payload, if present in a message, is the last payload in the message. Often, it is the only payload in the message. This rule is a critical property for securing IP packets, therefore, it is strictly obeyed.

81. EAP and Configuration Payloads are not applicable to our pre-decided configuration of controller-embedded system pairs.

82. The IKEv2 RFC states the IKEv2 is a security protocol and one of its major functions is to allow only authorized parties to successfully complete establishment of SAs. So a particular implementation may be configured with any of a number of restrictions concerning algorithms and trusted authorities that will prevent universal interoperability. Since a full interoperability with computers is not aimed, but only an embedded system to its controller is aimed, we are inside the limits of the standard. IKEv2 is designed to permit minimal implementations that can interoperate with all compliant implementations. Our work is a compliant minimal implementation with AES as the only algorithm used for all functions.

83. To avoid probing, the initiator of an exchange is required to identify itself first, and usually is required to authenticate itself first. The initiator can, however, learn that the responder supports IKE and what cryptographic protocols it supports. The responder (or someone impersonating the responder) can probe the initiator not only for its identity, but using CERTREQ payloads may be able to determine what certificates the initiator is willing to use. These

are not threats in our work. The initiator is a PC and it can employ additional security mechanisms.

84. Repeated rekeying using CREATE_CHILD_SA without additional Diffie-Hellman exchanges leaves all SAs vulnerable to cryptanalysis of a single key or overrun of either endpoint. Implementers are expected to take note of this fact and set a limit on CREATE_CHILD_SA exchanges between exponentiations. This is a good security consideration and can be easily implemented in our version of the work.

85. It is assumed that all Diffie-Hellman exponents are erased from memory after use. The strength of all keys is limited by the size of the output of the negotiated prf function. For this reason, a prf function whose output is less than 128 bits must not be used with this protocol. AES-128 is used in our implementation, which observes this rule.

86. The security of this protocol is critically dependent on the randomness of the randomly chosen parameters. These should be generated by a strong random or properly seeded pseudo-random source. When using pre-shared keys, a critical consideration is how to assure the randomness of these secrets. The strongest practice is to ensure that any pre-shared key contain as much randomness as the strongest key being negotiated. Deriving a shared secret from a password, name, or other low-entropy source is not secure. Quality of randomness is in parallel with the quality of the pseudo-random generator in the embedded system. This is a critical issue that concerns the manufacturers of micro-controllers. Assuming the embedded system generates unbiased random numbers our implementation is secure.

87. If the messages of IKEv2 are long enough that IP-level fragmentation is necessary, it is possible that attackers could prevent the exchange from completing by exhausting the reassembly buffers. The above argument is correct but does not apply to our work. Because the IKEv2 messages of our

implementation have been simplified to minimum length by the removal of the SA negotiation.

At the end of the above considerations and decisions it is clear that the IKE messages and their construction computations are greatly simplified. The implementation of IPsec in embedded systems is now much easier. Our work involves simplification around the AES encryption algorithm which is used for all security function roles. Addition of any algorithm as a different choice for a function; for example AES for confidentiality and SHA-1 for integrity, would increase the memory space if software implementation is opted, or die area if hardware implementation is opted and in any case the computation overhead will increase too. The development of an implementation has also been simplified in terms of time, debugging, testing and complexity.

CHAPTER SIX
DEVELOPMENT AND TESTING PLATFORM OF PROTO-TYPING

Design methods for networked embedded systems fall into the general category of system-level design. A decision has to be made on which the design is going to be constructed on. That decision affects the cost and testing methods. An academic environment like ours is in parallel with the trend of limited resources, in embedded systems. The design, development, debugging and testing is all founded, upon a platform. The platform is made up of both the hardware and the software devices and tools that are used in reaching the ultimate goal of a research. In the choice of a platform, there are a number of decisions to be made in terms of both hardware and software. The goal of reaching a prototype embedded system, in which a minimal IPsec runs, is a tough challenge. The platform chosen decides whether the research is going to be successful or not. Therefore, utmost care must be taken and decisions made must be tested before a full scale platform is set up.

A platform is an abstract model that hides the details of a set of different possible implementations as lower level components. The platform; for example a family of microcomputers, emulated network and operating systems, allows developers of application designs to operate without detailed knowledge of the implementation (e.g. the specific application running on the embedded system). At the same time, it allows platform implementers to have an idea of the design costs.

With the above guidelines in mind, several decisions have been made, during our research. Arguments about the programming language, operating system, development environment, hardware configuration and the network set-up have been detailed below.

6.1 Options of a Development Platform and the Choices Made

There are critical decisions to be made while deciding the platform for the projects design space. Naturally there are several options to choose from. The first work is the study of IPsec implementations. A decision has to be made about which of the available IPsec implementations to examine. This is important because different parts of an IPsec implementation can be designed in different ways. The design might lead the software implementation to violate the rules of IPsec RFCs. There are commercial IPsec implementations claiming both full-compatability with IPsec and also suitable for embedded systems. The accelarator and processor hardware necessary to run the implementations raise doubt whether they are suitable for low capacity embedded systems. The decision is left to the reader to study the references (Fusion 2010), (Mocana 2010), (QuickSec 2009) and question the implementations. But, it is obvious that if a full IPsec implementation is feasible in embedded systems, then our work is even more feasible. The code of the implementations has to be studied for clues and hints; an insight cannot be gained for making the right decision about the platform to be used for development. The studied implementations are used to understand the techniques of coding IPsec features, on computers. But it should not be forgotten that implementations made for computers presume abundant availability of resources. This is not the case in resource stricken embedded systems. Therefore, a study of all available implementations is vital before choosing the key components of our own platform that will lead to the simplified solution.

There are many IPsec implementations integrated into commercial operating systems. Latest version of Microsoft, Unix and Linux operating systems all have IPsec either as an option or directly integrated in the operating system. Then there are the operating systems with semi-commercial or open source copyrighted features. To list a few; BSD Unix and Ubuntu Linux all have IPsec add-ons that can be utilized with some degree of complexity. There are also available IPsec packages as add-ons, both commercial and open source. These are mostly for computers but lately some versions for high end embedded systems have also appeared. The producers of those made for embedded systems claim full compatibility with the computer version. They also claim having a small footprint in the memory but no measure of other resource usage is provided. It is also a fact that these packages call for an accelerator to also be integrated into the system.

While there are a number of IPsec choices under Linux Operating systems, there are only limited choices under Windows Operating Systems. The IPsec used in Windows is embedded in it and it is a commercial product. The source codes of the commercial versions are not available. Thus, commercial versions have to be dropped out and the attention has to be turned to the un-copyrighted and open source candidates. Those that are open source but integrated into an operating system cannot be easily analyzed. The BSD version of IPsec suite was the first to be studied. The implementation looked very operating system specific. Furthermore, the latest version of IKE has not been fully implemented in BSD and proper documentation, is not available. The second IPsec implementation examined was the Ubuntu extension. It also lacked IKEv2 implementation and the IPsec add-on was not supported by the operating system implementers. For this reason a search for an open source IPsec implementations without an operating system involvement was launched. As there are not very few available, only the available were studied. These are FreeSwan (Freeswan, 2003), OpenSwan (Openswan, 2003), StrongSwan (Strongswan, 2004), openIKEv2 (Gregorio, 2005) and IKEv2 (Domagoj & Gros, 2006) of sourceforge.net. FreeSwan is out dated and it has been dropped out.

OpenSwan and the StrongSwan are known to have originated from the FreeSwan project but the StrongSwan has received a lot of attention and continued support, until recently. These IPsec implementations have their own design characteristics but they are all open sources and free to use in academic research. A closer look at the “Swans” revealed that IKEv2 is implemented in StrongSwan only. But StrongSwan is implemented in object oriented programming therefore implementation comparison is not possible, as object oriented programming is regarded as not suitable for embedded systems due to the large code generated by the compilers, execution speed and memory requirements (Sunil et al. 2010).

There are two other open source IKEv2 packets. They are openIKEv2 and IKEv2, both freely distributed. OpenIKEv2 is written in C++ and the same argument for StrongSwan is also true for this package. The only left alternative is the IKEv2 written in C programming language. Although some important features of IKEv2 protocol have not been coded in this package, the absent features are unrelated to embedded systems and our target environment. Furthermore, the language to be used in the implementation of the embedded system software code is also important. Here the C language matches the C Programming tools developed specially for embedded systems, like Keil-C Programming tool (Keil, 2011), which is also a teaching tool in our labs. The program IKEv2 is not object-oriented, causing the code to grow out of embedded system memory. The code is also readable and commented.

Further examining of the programming language showed that free C language tools are also abundant. Tools for computers and limited devices exist in terms of compilers and editors, for the C Language. The software development environment is necessary for tools like keeping version numbers, easy to use editors, linking of files, debugging and tracing errors etc. Therefore, this decision is also fundamental. The cost of the minimum configuration requirements for the computers to be used has to be considered, as well. Copyrighted material has a cost naturally, even if used for academic purposes. Plus, the costs of the computers that are needed to run the programs add up to a considerable sum. Therefore, keeping the total cost of the platform in mind is important, too. The above advantages resulted in C programming language to be chosen as the main tool for development. After developing and testing the written software would be ported to the embedded system; through the Keil-C environment, a much easier porting path than any other tool.

The C Language decision is supported by another fact. In chapter five, AES is introduced as the choice for integrity, encryption and the pseudo-random function implementations. In this way, all three functions are reached in one package. Implementing only AES greatly simplifies the implementation for embedded systems and greatly reduces resource demands because all of the encryption, integrity and pseudo-random functions can be obtained from just one hardware crypto-engine included in the micro-controller of the embedded system. Implementation of AES as a crypto-coprocessor in an embedded system microcontroller was thought to be cumbersome and burdensome. However, Feldhofer (Feldhofer & Wolkerstorfer, 2009) showed that this was not so. Hence choosing AES is the right decision. There is the second advantage of having many open source implementations around. In this way there is no requirement for devising our own AES but simply using a concise version that runs efficiently in CFB mode would reduce the development period.

Then there is the operating system of the development platform. Embedded systems usually do not have an operating system. But talking about an IP stack or IPsec for a system without an operating system is out of question. Therefore, the target embedded system is one that has a minimal operating system that supports the IP stack. If not, the stack has to be put in front of the embedded system. That means a front end protector for embedded systems that cannot support an operating system. The research on the IPsec implementations also revealed clues of operating systems that can be used. The chosen IKEv2 package was developed on Linux. There are compilers for computers and low capacity devices, under the Linux operating systems. There are various Linux versions from a wide range of researchers. Support and fast help is readily available. Therefore, Linux operating system is the third choice made in the decision line. More specifically, the popular Ubuntu Linux with abundant support and easy to use features is chosen as the operating system of the development platform. This decision is supported by the fact that Eclipse Development Platform is available for C Language projects on both Windows and Linux. The version on Linux is free of charge and readily available to academic projects, like ours.

After the above decisions, the hardware configurations have to be considered. These involve the controller of the embedded system which has to be a computer, the computer that will emulate the embedded system itself, during the early stages of development and testing. The existence of another computer that examines the encrypted traffic between the controller and the embedded system, and finally the network in between are also needed. To list the total hardware needed, a controller PC, an embedded system emulator PC, a network analyzer PC, an embedded system emulator board, a front end protector emulator board, a local area emulator active switch and cabling. In short, 3 PCs, 2 boards and an active switch are needed to complete the platform.

The final decision to complete the platform is the individual configuration of the computers and boards that will be used and the network that is going to put them on the same LAN. The micro-computers used are naturally the modern personal computers supplied to the academicians. Multiple cores, 4 GB RAM and abundant disk space is common in today's PCs. Although there is no limitation imposed on the controller PC or the PCs that emulate the embedded systems at the early stages, micro-computers from the Pentium 4 era are also adequate for the development and debugging. The starting set-up is shown in Figure 6.1.

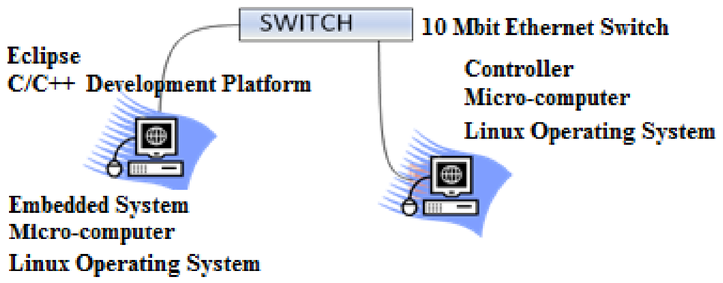


Figure 6.1 Development Platform of the Project

The early testing set-up involves a third PC that monitors the network, as shown in Figure 6.2. This is a network traffic analyzer that examines the traffic between the controller and the embedded system emulator. Further details of this set-up are given in the next chapter.

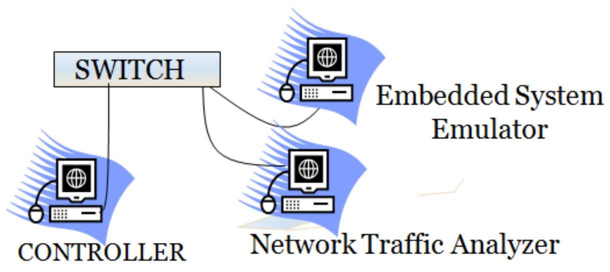


Figure 6.2 Debugging and Testing Set-Up of the Project

The architecture of the target embedded systems proves to be more critical. An embedded system with a network interface would suffice in the project, but the board that is used in medical devices is preferred; because the end product of this research has a chance to be tested in a real hospital environment. The embedded system board used as an emulator is preferred to have an Intel microcontroller on it. The first reason for the choice is the support and availability of the board, in our country. Importing or using an embedded system board that has no local support or service can cause problems during the research. The second reason is the long term use of the purchased boards in other projects. A board with the possibility of loading different operating systems and keeping as experimentation equipment later would be ideal for our academic purposes. Although the primary concern is wired Ethernet environment, wireless or hybrid IP applications can have the same benefits of our research. Hence a board with wired and wireless network capability is best choice that can be made. Therefore, the decision was to use two netbook boards with 1 GB RAM each and no hard disks. Only two 64 MB USB flash disks are enough for the platform of the project.

The network configuration of the platform is not a choice consideration but rather a preference. A full speed network of Gbits Ethernet is not very realistic in environments where embedded systems are used. 10/100 Mbits Ethernet LANs are still widely used and embedded systems mostly supports these speeds, because of their reduced cost. Similarly, our project utilizes an Ethernet switch of 10 and 100 Mbit ports.

6.2 The Configuration of the Chosen Platform

The final configuration of the chosen platform at the early development stage is given in Figure 6.3. All equipment is personal computers. There is no need for any limited capacity devices at this stage, because hitting resource limits like memory or processing power only hinders development. Therefore, reaching a working set and then optimizing and porting is a strategy, in projects like ours.

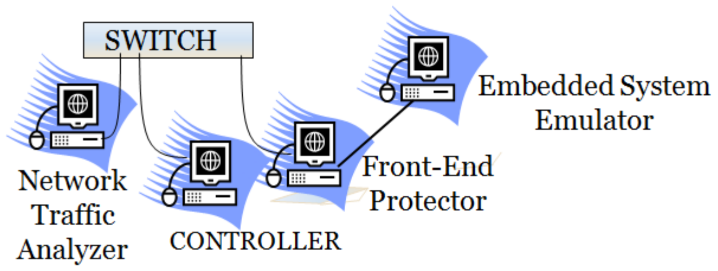


Figure 6.3 Testing Set-Up with The Protector

All of the computers have Ubuntu Linux operating systems. All of the computers have the Eclipse Development Platform supporting C Language programming, except the traffic analyzer. The traffic analyzer has Wireshark, software used for analyzing network traffic (Wireshark, 2011). Wireshark is open source software available on Linux operating systems.

The next strategy is porting the software to embedded system boards, after a working set of IKEv2 is reached. For this a platform of the form shown in Figure 6.4 is necessary. The front end protector is reached after a successful set is working on the embedded system. Once the simplified IKEv2 works on an embedded system it can be improved to function as a front end security box for an embedded system that cannot be altered. This set up is shown in Figure 6.5.

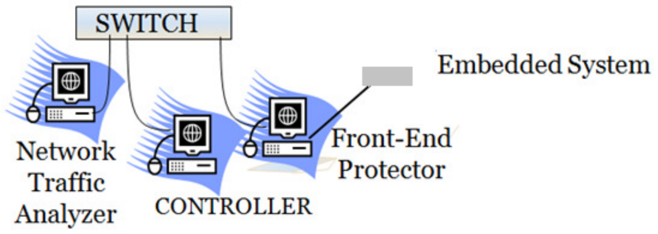


Figure 6.4 Porting to an Embedded System Board

6.3 The Hardware and Software Configuration of the Embedded System

The embedded systems used as an emulator and a front end protector are not of equal hardware configuration. The embedded system is a limited capacity device, while the front end protector can have good resources as it will reside outside the embedded system. But since it will increase the cost keeping its configuration is a good practice. This criterion would force the target IKEv2 implementation to be as small as possible so that it would fit to very small embedded system configurations as well. Such an effort would make the project to building a secure embedded system micro-kernel capable of networking.

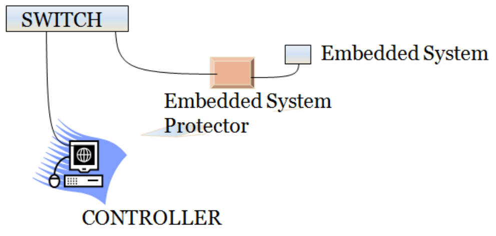


Figure 6.5 Final Prototype Set-Up of the Research

Designing micro-kernels is a special topic, which focuses deliberately on optimizing the whole kernel running an embedded system. Our work helps that goal by reducing the IP security stack, through compatible simplification of IPsec. It is exactly at this point where the security of an embedded system is weakened, by focusing on a micro-kernel and not implementing a standard based security scheme.

The embedded systems used in our work are deliberately chosen to have the similar microprocessor architecture, for making the porting process easier. Choosing a different architecture or manufacturer for the embedded systems merely prolong the development process. Porting to different platforms is not impossible but only more laborious. Therefore the embedded system boards are boards that have Intel Atom microprocessors. These microprocessors have been manufactured by Intel for the embedded system market and they are aggressively priced. The developed software does not need to be compiled for a second time if the same operating system used in the micro-computers is used. But the operating systems of embedded systems targeted by our work are not the ones which allow full Ubuntu Linux to reside on them. Those systems that can accommodate only a small Linux version are the target.

To find the smallest possible Linux for the development platform, many have been studied. Size, distribution, support and open source properties have been used as the decision criteria. Finally, the distribution used in our platform is Tiny Core Linux which has a 7 MB non graphic user interface version or a 12 MB version with a simple graphic user interface of the operating system. This amount is small enough to fit in maximum 16 MB bootable flash disk together with the compiler and the developed implementation.

As a result, three computers with two core Intel processors, each having 2 GB of RAM, loaded with Ubuntu Linux operating systems and Eclipse Development platform with C language support constitute the starting phase of the lab work. The two computers are later replaced with two netbook boards having Intel Atom processors, 1 GB RAM, Tiny Core Linux and gcc as C Language compiler. A table summarizing the development and prototyping platform choices is given below.

Table 6.1 Development and Prototyping Platforms.

Feature	**Personal Computers**	**Embedded Systems**
Processor	Intel Core2	Intel Atom
Operating System	Linux	Tiny Core Linux
Development Platform	Eclipse Environment	None
Programming Language	C Language	C Language
Compiler	gcc	gcc
Secondary Memory	Hard Disk	Flash Disk
Network Traffic Analyzer	Wireshark	None
Controlled Hardware	None	DC Motor & Leds

CHAPTER SEVEN
LAB WORK AND THE RESULTING PROTOTYPE

The development of a target prototype -a secure embedded system- is the ultimate goal. The lab-work consists of planned phases, based on the previous phase results. Until reaching the goal, the results of intermediary lab-work are needed before obtaining a simplified IKEv2 for embedded systems. The secondary goal is the security of the embedded systems that cannot be upgraded or altered. Some expensive systems contain old, fixed-configuration embedded systems that have insufficient resources that cannot be upgraded. For instance, the old MRI devices of hospitals are expensive and cannot be altered. A small device system similar to an embedded system, acting as a front end protector for the insecure MRI device is needed to fulfill the missing security. The implied device is an embedded system protector; which is normally known as a "crypto-box" device, in cryptology. Crypto-boxes in fact are limited device that simply encrypt and decrypt messages between two communicating partners. Our protector not only does the same, but it also hides the embedded system behind itself, away from the eyes of the hackers on the local network.

Therefore, lab work consists of two parts. The first one is working on an embedded system that fulfils the IKEv2 of IPsec and continues the communication in encrypted form. The second is creating a front end protector in the form of a crypto-box that can be used to protect those systems that no security upgrades can be carried on. Although, it appears as two parts the core work is the same. The core work is the creation of an agent that carries out a reduced IKEv2 and a fixed version of IPsec. By fixed it is meant that both agents are expected to run in native-tunneling mode using only AES as integrity, confidentiality and pseudorandom functions.

Obtaining such a front-end protector is not difficult, once a secure prototype is reached. The work is the same except an additional secure IP intermediation is needed for the protector. It can be summarized as a network address translator but it is more than that. This goal is also pursued in our work.

A starting test platform is set up as shown in Figure 6.1 of the previous chapter. In fact, other platforms had also been set up for early testing of the platforms themselves; but they were discontinued. The IPsec and IKEv2 simplifications that will lead to a reduced implementation in a prototype have been prepared on paper, first. This involved the redrafting of the organization of IPsec described in Chapter 5. While redrafting the organization, the study and simplification of the RFC rules to create a DOI for embedded systems has to be carried out in parallel. Having finished the theoretical work the newly designed IPsec IKEv2 has been simulated on a cryptography tool.

The simplification of IPsec IKEv2 has been described in detail, in Chapter 5. The first work of drafting a Domain Of Interpretation for Embedded Systems (DOIES) obtained in Chapter 5 involves the study of the overall IPsec architecture, first. The study is focused on the three databases used in IPsec, because database applications are prohibitively resource demanding for embedded systems. The three databases used in IPsec; namely PAD (Peer Authentication Database), SPD (Security Policy Database) and SAD (Secure Association Database) are redundant in our application. The details of these databases are not discussed here. But suffice it to say that they can be omitted from embedded systems and only kept in controllers for book-keeping, in terms of user interfaces used. Considering the fact that indented communication is only among two partners - the controller and the embedded system- there are not too many peers, polices or associations to consider, in our case. The controller and embedded system are not total strangers to each other; as it is assumed in IPsec. That results in a big simplification as there is a limited number of info (i.e. SAs, IDs, SPIs and addresses etc.) to keep track of. Hence, data of a few lines long do not need fully fledged user interfaces or databases. Negotiating different security crypto and hash combinations is also out of question as it is fixed to one single function of AES. Therefore redundant negotiations and databases can be safely eliminated from our design.

Before starting the coding of the new IKEv2, taking the newly drafted Domain Of Interpretation for Embedded Systems (DOIES) obtained in Chapter 5, the simplified

IKEv2 has to be designed and tested on a preferably graphical tool. A design tool called Cryptool (Cryptool, 2011) is suitable to implement and test the simplified IKEv2 exchange. This tool has a strong library of already verified functions, such as encryption algorithms, attack simulator, result comparators etc. There is also a graphic interface for the encryption and creation of messages communicated. If used correctly a full description of the new IKEv2 exchange steps can be represented and tested before coding the overall design. Such a representation is shown in Figure 7.1, where the Diffie Hellman exchange of public keys by the partners those results in the calculation of SKEYSEED of IPsec. SKEYSEED is a key derived independently by the tow communicating partners which is in turn used as a seed for obtaining other keys for integrity, pseudo-random functions and authentication procedures, one for each direction.

As it can be seen from Figure 7.1, the two results obtained by A and B are compared and a decision is made if an equal result has been obtained. In AES, a change in one bit results in change of numerous bits spoiling the overall result. Thus

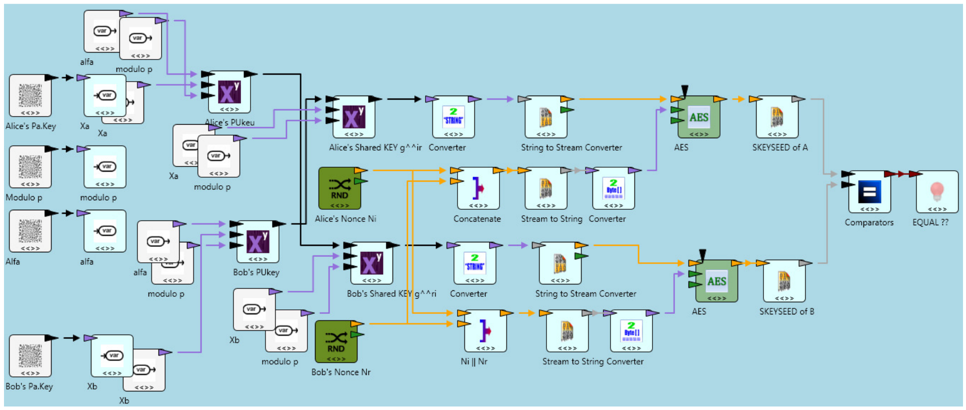


Figure 7.1 Representation of IKEv2 Exchange And Calculations in Cryptool

a total agreement on all 128 bits obtained is a must. The final graphical design of obtaining all the keys by using the SKEYSEED is much more complicated and even graphically difficult to follow. Nevertheless, a check over the design can be made by using the same values in coding and matching the resultant keys generated.

The testing of the results of the designed exchange is critical because once coded and run, it is difficult to detect erroneous results and pinpoint the cause of the error. The final values of the keys obtained depend on a chain of exchanges and calculation results. A mismatching key can be reached because of a bit error during communication or calculations. Or some functions are not well designed and produce a non standard result. Running the same function, e.g. AES, with the same inputs and getting the expected standard result is also vital at this stage. Once confident in the results the design can then be coded.

The planned coding phases of reaching a secured embedded system prototype from the above simplifications and designs are given below, in detail. The development started on a microcomputer platform as described in sections 7.1 and 7.2. The obtained resulting software was then downgraded to an embedded system level, which are described. Finally, real world implementation of the prototyped device testing and results are given in sections 7.4 and 7.5.

7.1 Pre-Development Phase on Microcomputers

The first phase starts with the setting up of the computers and loading their operating systems. Two microcomputers with equal hardware resources have been set up as shown in Figure 7.2. The hardware configuration of these micro-computers has been given as dual cored CPUs. It is obvious that the resources of the micro-computers are abundant compared to limited capacity embedded systems; with processing power over 2 GHz and memory capacity not less than 2 GB. This is because these two computers are for developing the IKEv2 and not the target devices. The Linux operating systems are Ubuntu Linux, with full networking support packages. The Linux operating systems of both microcomputers have full library support connection to the Internet. The academic versions of Ubuntu Version 9.10 at the start and regular updates to version 10.10 are sufficient to support the necessary development environment of the work. Part of the hardware is financed by Dokuz Eylül Üniversitesi BAP Project Number 2009.KB.FEN.044.

Next is the loading of the development environment and the programming tools such as compilers and editors, which finalize the set-up of the first phase platform. The Eclipse Development Platform with its C/C++ programming add-ons are loaded and default setting are converted into correct, local usage settings for the start.

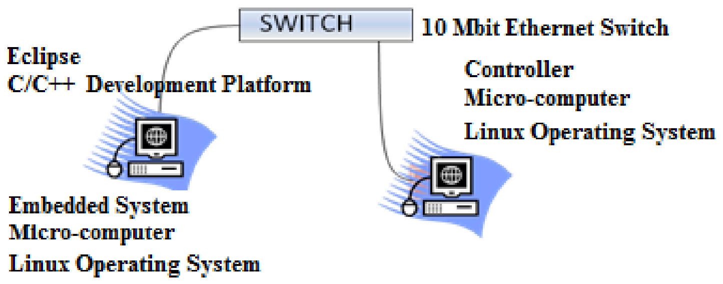


Figure 7.2 Starting Hardware Set-Up of Lab Work

The final step is the setting up of a network that connects two micro-computers to each other. In our case two computers over a network emulated by an Ethernet switch have been set-up (Figure 7.2). The Ethernet speed of the network is intentionally set to 10 Mbits/sec, to mimic a low speed local area network. This is expected to serve as a check on the response time of exchanges; to expose any delays resulting from encryption or message preparation delays, during the exchanges.

The hardware set up and fine tuning is not a negligible effort. While preparing the platform, it has to be remembered that the designed code will need to be ported to run on embedded system. It would be ideal that the code is compiled on this platform and simply carried to the target embedded system. Therefore minimal configurations are necessary at every step.

After the set up of the platform the coding begins. The simplified standard that is reflected by the tested design on Cryptool is coded step by step. The coding of global and local parameters is important if the overall software is going to be modular; i.e. partly transferable to other applications. Another point is that the intermediary results have to be tested and compared before the overall program is finished. For this reason the testing of the software is carried out module by module with an overall testing at the end, as described in the next section.

7.2 Development and Testing on Micro-computers

While developing the software a limited number of operating system or compiler crashes were witnessed. There were also short cut key lock-ups, hardware freezes at the start, but later these adverse conditions dissolved into a minimum by making further slight fine tuning. In summary, two solutions or prototypes are obtained at the end of this period.

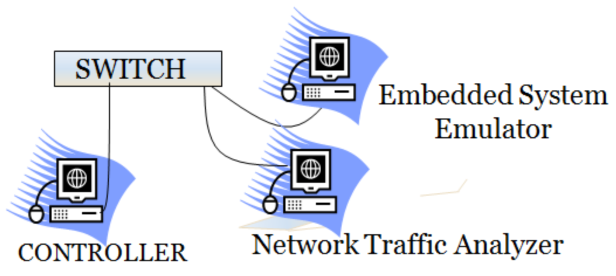


Figure 7.3 Analyzing Traffic between the Controller and the Embedded System.

The first result is a solution for high capacity embedded devices that can accommodate IPsec's IKEv2. The Embedded System Emulator in Figure 7.3 is the prototype of this work. The IKEv2 running on the controller and the embedded system are debugged first. The first step is to get a program running free of errors. The second step is controlling the exchanged public keys. After correct exchange of messages the third step is obtaining the session keys. The resultant keys obtained must be the same in both sides. The keys have to match in all of the 128 bits. Once the same keys are obtained the testing step can begin.

The developed IKEv2 is tested by letting an exchange take place and observing the communication between the controller and the embedded system, using the traffic analyzer. Any unencrypted message is seen on the packet analyzer screen of the observer, while encrypted characters cannot be. If a 6 messages exchange completes successfully, then a secure channel has been established. Then by taking an unencrypted file and sending it over TCP or UDP, after encrypting it with the obtained session key forms the first part of testing. The second part consists of the recipient decrypting the file using the obtained session key. The third part is the examining of the files exchanged for errors. If the exchanged files have correctly reached to either side then the secure channel has been established. The fourth part of testing is analyzing the traffic between the communicators to see if any of the exchanged files are readable. If not then it can be concluded that a secure tunnel has been established and that the embedded system has been secured.

Apart from developing the IKEv2 exchange the IPsec packets have to be also constructed an exchanged. That is to say the insertion of IPsec aware UDP and TCP communication has to be also developed, because the testing is carried between two real systems. The socket programming, the ports to be used, the IP address resolution that were discussed and planned in the previous chapters have to be implemented.

After successfully implementing the IKEv2 exchange and establishment of a secure channel/tunnel there is the porting work onto the embedded systems. But before going to compacting or reducing the software to fit in the embedded systems; the case of implementing the software for the front-end protector can be taken up. This is a better strategy because the simplification of the software can be done in parallel for both the embedded system and the protector since most of the code is the same for both.

Obviously, the work for the protector depends on the previous code development. Since the protector acts as an intermediary between the controller and the embedded system, the IKEv2 exchange carried out by the embedded system has to be moved over to the protector. This is easy and means the exchange and key generation role will be carried out exactly as before by the protector. After a secure channel is established between the controller and the protector; communication between embedded system and the controller can begin. This set-up is shown in Figure 7.4. Observe that the connection of the embedded system has been removed from the switch and moved to the network interface card (NIC) of the protector. Hence the protector must have two NICs, two sub-nets and be able to route the IP packets from one to another. That is to say, monitor the first NIC connected to the controller and watch out if there is any message to the embedded system; which comes from a specific TCP or UDP port. Relay the message of the controller to the embedded system on a specific TCP or UDP port that the embedded system is known to be listening. Of course the message has to be decrypted first.

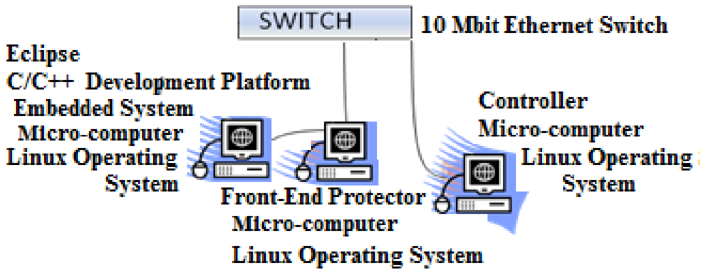


Figure 7.4 Testing of Front-End Protector

Similarly, any message from the embedded system to the controller on a pre-defined TCP or UDP port is encrypted and sent. This work is the implementation of a front-end protector (crypto-box) for non-upgradable systems. The resulting work has to be ported on to a low capacity embedded system. Both work need to be tested thoroughly and updated according to the results. This is most of the work in the development and testing period. These tests have been finished with positive results to make the testing and debugging simpler and easier on embedded systems.

Table 7.1 Comparison of the original standard and our implementation.

Feature	IPsec	Ours
Protocols offered	AH, ESP	ESP
Implementation methods	BITS and BITW	BITS or BITW
Implementations offered	Gateway-to-Gateway, Endpoint-to-Endpoint, Endpoint-to-Gateway	Endpoint-to-Endpoint
SA negotiation	Required	None
Databases Required	SAD, SPD, PAD	None
Modes Offered	Transporting, Tunneling	Tunneling
Confidentiality Algorithms	DES, 3DES, RC5, IDEA, CAST, BLOWFISH, AES_CBC, AES_CTR	AES_CBC
Integrity Algorithms	MD5, SHA-1, DES_MAC, KPDK_ MD5, AES_128_XCBC	AES_128_XCBC
Pseudo-random Functions	MD5, SHA-1, TIGER, AES_128_XCBC	AES_128_XCBC
Management Techniques	Manual and Automatic	Manual
Key Management GUI	Required	Optional
Key Length	Variable	Constant
Authentication Methods	RSA Digital Signature, Shared Key Message, DSS Digital Signature	Shared Key Message
Payload Sizes	Large, dependent on SA negotiations	Smaller
Communicated Messages	Large, dependent on SA negotiations	Smaller
Power Consumption	High	Less

The comparison between the original IPsec standard and our implementation is given in Table 7.1. The complication of the IPsec standard is obvious from the multiple alternatives in all of the offered features and the variable key length property. Observe how simplified our implementation is in the payload sizes and communicated messages. The communicated messages are small both in size and in numbers, during the establishment of a secure channel. As a result, the overhead incurred in terms of accommodated software, computations, and message transfer is much smaller compared to the overhead caused by the complicated IPsec standard.

The increase in power consumption due to security additions had been explained in Chapter 3. It is obvious that IPsec uses a lot of power due to its databases, indeterminate number of negotiations during the establishment of the secure channel. Our simplified version uses much less power for these chores as there are no prolonged choice negotiations or databases. Otherwise, the power used for the encryption of the followed communication is the same in both. Table 7.1 shows that overall power consumption of our proposal is less than that of IPsec.

7.3 Development and Testing on Embedded Systems

The previous work on the micro-computers serves as the basis of the work on the embedded systems. By keeping the architectures equivalent the development and testing of implementations simplify greatly. The porting reduces to compilation of the source code on the target embedded systems. This is only necessary to see if the libraries are existent and consistent on the target embedded system as well. Otherwise even compilation is not necessary for small program subroutines. For these reasons the debugging of the developed implementation on the embedded systems are far easier than the debugging carried out on the micro-computer platform. It is true that some unexpected error due to limited capacity do occur on the embedded systems, but they are obvious errors openly declared by compiler messages; which can be fixed easily.

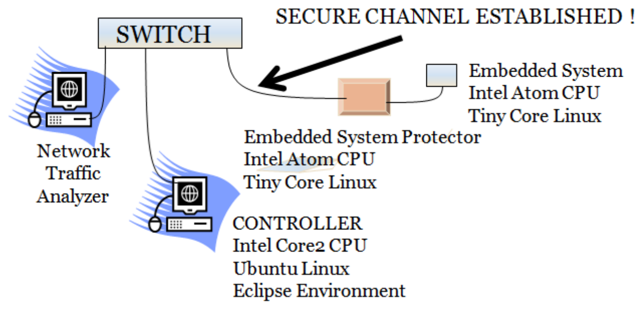


Figure 7.5 Successful Implementation of IKEv2 On Prototypes.

The testing of the two set-ups shown in Figure 7.4 and Figure 7.5 are identical. Since the results of the tests carried out on the micro-computer platform all that is needed is their comparison with the embedded system platform. Results showing that no information is readable on the wire and the equivalence of encrypted messages on both platforms show that the secure channel has been established.

This marks the end of the implementation of the IPsec key exchange and the authentication steps, in the lab. It also means that the step of porting the solution to the front-end protector configuration has been finished, too. The security additions not suitable for a low capacity embedded system are thus implemented on a mediator as shown in Figure 7.5, proving that the original goal of securing embedded systems can be achieved as claimed. More simplification and optimization rounds are necessary for an even more reduced version of IPsec for embedded systems.

7.4 Testing of the Embedded System Prototype

After implementing IKEv2, on the test platform the secure channel established has to be tested for secure exchange of information over other IP ports of different applications. A Table showing the ports used for testing is given in Table 7.2. The controller has two pairs of receive and transmit ports; one for a TCP and one for a UDP application. Similarly the embedded system also has two pairs. The test is carried out by entering the ports and IP addresses into the application manually, at first. The traffic between the controller and the embedded system is monitored at the

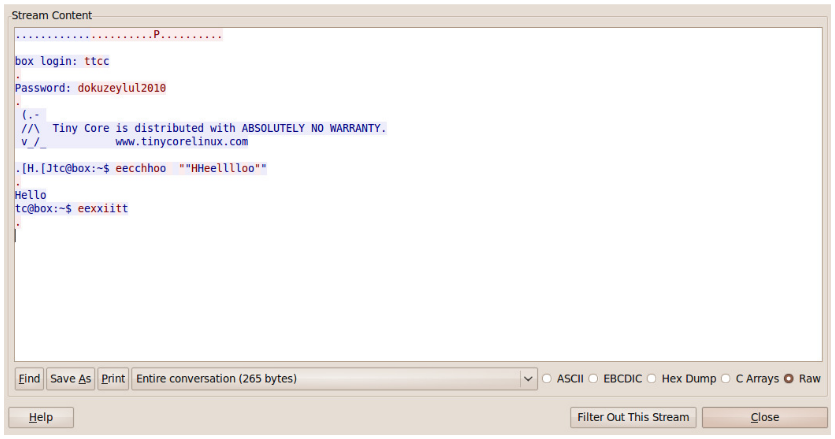
Table 7.2 IP Port Numbers for Testing the Prototypes.

CONTROLLER				**PROTECTOR**		
UDP	Receive	8800	←←←←←	UDP	Send	8800
UDP	Send	8801	→→→→→	UDP	Receive	8801
TCP	Receive	8880	←←←←←	TCP	Send	8880
TCP	Send	8881	→→→→→	TCP	Receive	8881
EMBEDDED SYSTEM				**PROTECTOR**		
UDP	Receive	8700	←←←←←	UDP	Send	8700
UDP	Send	8701	→→→→→	UDP	Receive	8701
TCP	Receive	8770	←←←←←	TCP	Send	8770
TCP	Send	8771	→→→→→	TCP	Receive	8771

Traffic Analyzer. The application drives sample medical equipment, where a DC motor and indicators showing the state of the equipment are controlled. The control signals are sent in encrypted form, thus even an insider who has identical equipment cannot determine the control signals; because the encryption key is obtained between the controller and the embedded system. Without the session key, the insider cannot decrypt the communication to obtain the control signals. The packets collected by the analyzer do not make a meaning even if stored; because the revelation of a latest key is not useful to decrypt the store history. Thus the meaningless, unprintable characters on the packet display of the traffic analyzer prove the security of the communication.

Testing also involves injecting packets into the communication. These bogus packets are identified by the controller easily as it is using anti-replay, SPI, Message ID features. Similarly, the embedded system identifies bogus messages and raises a pre-determined alarm or drops the communication completely. While testing the embedded system prototype, all IP, IPsec stack (BITS) and IKEv2 operations are executed within the embedded system. The response time and the performance of the embedded system show no unsatisfactory results.

Results of testing without any security prove that the whole traffic is readable, by the use of the Wireshark packet analyzer. Figure 7.6 shows the network traffic of a telnet session between the embedded system and the controller. The contents of the packet stream can be skimmed to only find out the user ID and its typed password displayed by the analyzer.



```
.....................P..........

box login: ttcc
.
Password: dokuzeylul2010
.
 (.-
 //\  Tiny Core is distributed with ABSOLUTELY NO WARRANTY.
 v_/_           www.tinycorelinux.com

.[H.[Jtc@box:~$ eecchhoo  ""HHeellllloo""
.
Hello
tc@box:~$ eexxiitt
.
```

Entire conversation (265 bytes)

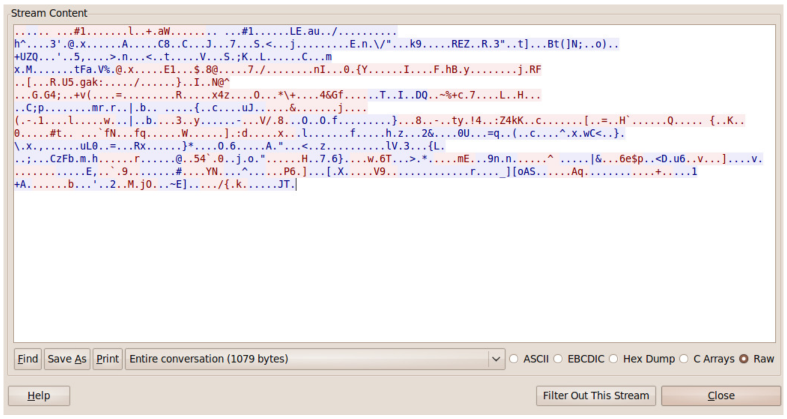
ASCII EBCDIC Hex Dump C Arrays [✓ Raw]

Figure 7.6 Network Traffic without implementing security.

7.5 Testing of the Embedded System Protector Prototype

Table 7.1 also shows the ports used by the front-end protector. Naturally, the protector - or crypto-box - has four pairs of ports. One pair is for the communication with the controller and the other is for the communication with the embedded system, as shown in Figure 7.5. The protector has two NICs. The NIC connected to the embedded system is assumed to have a short Ethernet cable connected to the embedded system. The connection between the crypto-box and the embedded system is assumed to be secure. The ports between the crypto-box and the controller are monitored like in the previous section. The testing and results are the same as expected.

The only remaining work is tunneling software, on the controller and the crypto-box that monitors all of the port traffic and act as IPsec BITW mediator. Tunneling software is merely a client-server software; if IPSec BITS mode is not implemented inside the operating system. A client-server software has been developed where the protector (server) listens for controller (client) calls. If a call comes the tunnel is set up where all traffic is forwarded to a port that uses the IKEv2 and IPsec software developed. Thus our work contains the development of a client-server tunneling software which implements the above developed IKEv2 and IPsec features.



```
...... ...#1.......l..+.aW....... ...#1......LE.au../.........
h^....3'.@.x......A.....C8..C...J...7...S.<...j.........E.n.\/"...k9.....REZ..R.3"..t]...Bt(]N;..o)..
+UZQ...'..5,....>.n...<..t.....V...S.;K..L......C...m
x.M.......tFa.V%.@.x.....E1...$.8@.....7./........nI...0.{Y......I....F.hB.y........j.RF
..[...R.U5.gak:...../......}..I..N@^
...G.G4;..+v(....=.........R.....x4z....O...*\+....4&Gf.......T..I..DQ..~%+c.7....L..H...
..C;p.........mr.r..|.b.. .....{..c....uJ.......&.......j....
(.-.1....l.....w...|..b....3..y......-...V/.8...O..O.f.........}...8..-..ty.!4..:Z4kK..c........[..=..H`......Q..... {..K..
0.....#t.. ...`fN...fq......W......].:d.....x...l.......f.....h.z...2&....0U...=q..(..c....^.x.wC<..}.
\.x.,......uL0..=...Rx......}*....O.6.....A."...<..z..........lV.3...{L.
..;...CzFb.m.h......r......@..54`.0..j.o."......H..7.6}....w.6T...>.*.....mE...9n.n......^ .....|&...6e$p..<D.u6..v...]....v.
.............E,..`.9........#....YN....^......P6.]...[.X.....V9...............r...._][oAS......Aq..............+.....1
+A.......b...'..2..M.jO...~E]...../{.k......JT.
```

Entire conversation (1079 bytes) ASCII EBCDIC Hex Dump C Arrays [✓ Raw]

Figure 7.7 Network Traffic with security implemented.

The same login page of the telnet session tried between the embedded system and the controller is shown in Figure 7.7. The result is a scrambled page with meaningless, unprintable characters. This is because the whole telnet session is encrypted; therefore the analyzer has no way of interpreting the stream into meaningful characters. This is a proof of the security provided to a insecure embedded system.

CHAPTER EIGHT
CONCLUSION

8.1 Conclusion

In order to make the embedded systems more reliable in general, a standard that covers them all which increases their security is imperative. This standard cannot be made by a private group of researchers like us. Only, the already established standards can be used to derive a subset for the embedded systems. This subset can act as a first step to reach a general standard. Our work is such an attempt of forming the basis of a joint effort towards more secure embedded devices.

The contributions of this work to the introduction of security to low-resourced embedded systems can be summarized as:

- Simplification of IPsec standards into a consistent subset that is feasible in embedded systems,
- Simplification of IPsec into an invariable ESP mode,
- Simplification of IPsec into an invariable tunneling mode,
- Simplification of IPsec databases and their management,
- Simplification of IPsec implementation into an invariable host-to-host mode,
- Simplification of IKEv2 payloads,
- Simplification of IKEv2 confidentiality, integrity and pseudo-random functions into AES.

At the end of the above contributions, a prototype of the simplified IKEv2 and IPsec features was implemented, in an embedded system. A prototype embedded system, acting as a protector in front of a vulnerable embedded device, using the simplified IKEv2 and IPsec features was also implemented.

This effort has brought increased confidentiality, reliability and integrity to embedded systems, which previously had none or very little. Traffic analysis proves that previously exchanged clear text information, is no longer readable after its encryption based on a strong standard. The prototype obtained in our work proves that complex and immaculate standards prepared previously for computers, can be simplified to fit in embedded systems. Instead of designing a new but debated security standard, using a subset based on internationally accepted standards has a better chance for gaining support, in the community. By following our work, it is possible to obtain a common ground for the security and reliability of embedded systems, in general.

8.2 Future Work

Our prototypes need to be optimized further. The libraries of the tools used for developing a target implementation can be simplified or reduced, as well. Such simplification and optimization would make the obtained end-product to address embedded systems of smaller amount of resources. Further simplification can be reached if the repeating fields in payloads, and thus the payloads themselves can be simplified or reduced.

Another future work is not further simplification but increasing the alternatives of supported encryption or integrity functions. Still the negotiation can be simpler by pre-defining alternatives into arranged pairs; like only AES as alternative 1, AES and SHA-1 as alternative 2 etc. Upon receiving a challenge with an additional parameter 2, an embedded system knows that it has to use AES and SHA-1.

The optimization work, however, can only be relinquished if the embedded systems grow into the computers of last decade. The fast growth of the sizes of the resources inside the embedded systems cannot be denied, but optimization will continue and only bring the time of acceptance of our implementation at a closer date.

REFERENCES

Anderson, R.J. (1993), Why cryptosystems fail, In *Proceedings of ACM CSS'93,* ACM Press, 215–217.

Anderson, R., Kuhn, M. (1996), Tamper resistance-a cautionary note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, 1–11.

Anderson, R. J., Kuhn, M.G. (1997), Low cost attacks on tamper resistant devices, In *Proceedings of the Fifth International Security Protocols Conference*, Lecture Notes on Computer Science, (1361), Springer-Verlag, 125–136.

Anderson, R. J. (2001), Security Engineering: A Guide to Building Dependable Distributed Systems (27-35). U.S.A.: John Wiley & Sons.

Barr, M., Massa, A. (2006). In *Programming Embedded Systems* (2nd ed.) (13-15), U.S.A: O'Reily.

Boneh, D., DeMillo, R. A., Lipton, R. J. (2001), On the importance of eliminating errors in cryptographic computations. In Journal of Cryptology: *The Journal of the International Association for Cryptologic Research*, (14), 101–119.

Brodsky, J. (2009), Attaining non-disruptive SCADA security upgrades with a systematic, incremental change strategy. In *SCADA and Control Systems Security Summit.*

Burke, J., McDonald, J., Austin, T. (2000), Architectural support for fast symmetric-key cryptography, In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, 178–189.

Carman, D.W., Kruus, P.S., Matt, B.J. (2000), Constraints and approaches for distributed sensor network security, In *NAI Labs: Technical report*, (2000-110), Retrieved on May 4, 2011, from http://www.cs.umbc.edu/courses/graduate/CMSC691A/Spring04/papers/nailabs_report_00-010_final.pdf.

Cryptool (2011), Cryptool for Cryptography and Cryptanalysis, Retrieved on May 5, 2011, from http://www.cryptool.org.

Daswani, N., Boneh, D. (1999), Experimenting with electronic commerce on the PalmPilot, In *Proceedings of the Third International Conference on Financial Cryptography*, 1–16.

DIES, Distributed and Embedded Systems Group, Faculty of EEMCS, University of Twente, The Netherlands, Retrieved May 4, 2011, from, http://dies.ewi.utwente.nl/ twente.

Diffie, W., Hellman, M. (1976), Multiuser Cryptographic Techniques, In *IEEE Transactions on Information Theory*, (November).

Dolev, D., Yao, A.C. (1983), On the security of public key protocols, In *IEEE Transactions on Information Theory*, (29), 198-208.

Domagoj, F., Gros, G. (2006), IKEv2 Project, Retrieved on May 5, 2011, from, http://www.sourceforege.net/projects/ikev2/.

Doraswamy, N., Harkins D. (2003), IPSec: The new standard for the Internet, Intranets, and Virtual Private Networks (2nd ed.) (17-22), U.S.A.: Prentice Hall.

EFF (2004), U.S. v. ElcomSoft and Sklyarov FAQ, Retrieved on April 25, 2011, from, http://www.eff.org/IP/ DMCA/US_v_Elcomsoft/us_v_sklyarov_faq.html.

ESSG, Embedded System Security Group, University of Massachusetts Amherst, USA, Retrieved May 5, 2011, from, http://vcsg.ecs.umass.edu/essg/home.html.

EmSeC, UCLA Embedded Security Group, University of California at Los Angeles, USA, Retrieved May 4, 2011, from, http://www.emsec.ee.ucla.edu/.

Feldhofer, M., Wolkerstorfer, J. (2009), Hardware Implementation of Symmetric Algorithms for RFID Security, In *RFID Security: Techniques, Protocols and System-on-Chip Design,* (3) (373-415), U.S.A.: Springer.

Frankel, S. (2001), In *Demystifying the IPsec puzzle* (1st ed.) (12-13, 41-43, 77-78, 87-90), US.A: Artech House.

Frankel, S., Herbert, H. (2003). The AES XCBC-MAC-96 Algorithm And Its Use With IPsec, In *RFC 3566*.

Freeswan, (2003), The Linux Freeswan Project, Retrieved on May 5, 2011, from, http://www.freeswan.org/.

Fusion Embedded IPsec (2010), Unicoi Systems, Retrieved on May 23, 2011, from http://www.unicoi.com/fusion_secure/fusion_ipsec.htm

Gorman, S., Dreazen, Y. J., Cole, A. (December 17, 2009), Insurgents Hack U.S.Drones, *Wall Street Journal*, Retrieved on May 5, 2011, from http://online.wsj.com/article/SB126102247889095011.html.

Gunther, S.H., Binns, F., Carmean, D.M., Hall, J.C. (2001), Managing the impact of increasing microprocessor power consumption, In *Intel Journal of Technology*, (Q1: 9).

Gutmann, P. (2002), Lessons learned in implementing and deploying crypto software, In *Proceedings of the 11th USENIX Security Symposium*, 315 –325.

Gregorio, U. (2005), Strongswan Project, Retrieved on May 5, 2011, from, http://www.sourceforege.net/projects/openikev2/.

Heath, S. (1995). Embedded Systems Design. (2nd ed.) (1-4), U.S.A.: Newnes.

Huang, A. (2003), Keeping secrets in hardware: The Microsoft X-Box Case Study, In *Revised Papers from the Fourth International Workshop on Cryptographic Hardware and Embedded Systems*, 213–227.

Hwang, D. D., Schaumont, P., Yang, S., Verbauwhede, I. (2006), Multi-level Design Validation in a Secure Embedded System; IEEE (55), 1380-1390.

Kaufman, C. (2005), Internet Key Exchange (IKEv2) Protocol, In *RFC 4306*, IETF.

Kent, S., Seo, K. (2005), Security Architecture for the Internet Protocol, In *RFC 4301*, IETF.

Kent, S. (1998), Security Architecture for the Internet Protocol, In *RFC 2401*, IETF.

Kent, S., (2005), IP Encapsulating Security Payload, In *RFC 4303*, IETF.

Keil (2011), Keil C Embedded Development Tool, Retrieved on May 5, 2011, from, http://www.keil.com.

Kingpin, (2000), Attacks on and countermeasures for USB hardware token device, In *Proceedings of the Fifth Nordic Workshop on Secure IT Systems Encouraging Co-operation*, 135–151.

Kocher, P., Freier, A.O., Karlton, P. (1996), The SSL protocol Version 3.0 specification, IETF.

Kocher, P.C. (1996), Timing attacks on implementations of Diffie-Hellman RSA DSS and other systems, In *Proceedings of CRYPTO '96, Lecture Notes in Computer Science*, (1109) 104–113.

Kocher, P., Ravi, S., Lee, R., McGraw, G., Raghunathan, A. (2004), Security as a new dimension in embedded system design. *In Proceedings of the 41st Annual Conference on Design Automation*, 753–760.

Kömmerling, O., Kuhn, M. G. (1999), Design principles for tamper-resistant smartcard processors, In *Proceedings of the USENIX Workshop on Smartcard Technology* (Smartcard '99), 9–20.

Leyden, J. (2009). Old worm learns Conficker tricks. *The Register*. http://www.theregister.co.uk/2009/04/06/old_worm_adopts_conficker_tricks/.

Liedtke, J. (1996), Towards Real Microkernels, In *Communications of the ACM*, 39 (9), 70–77.

Maher, D.P. (1997), Fault induction attacks, tamper resistance, and hostile reverse engineering in perspective, In *Proceedings of the First International Conference on Financial Cryptography*, 109–122.

Martin, T., Hsiao, M., Ha, D., Krishnaswami, J. (2004), Denial-of-Service attacks on battery-powered mobile computers, In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications* (PerCom'04), 309.

Mazidi, M.A., Mazidi, J. G., McKinley, R.D. (2006), The 8051 Microcontroller and Embedded Systems (23-25). USA: Prentice Hall (Pearson).

Mocana (2010), Medical Devices Hacked, *Mocana Corporation*, Retrieved on May 5, 2011, from, http://mocana.com/blog/2010/04/08/medical-devices-hacked/

Moore, S., Anderson, R., Cunningham, P., Mullins, R., Taylor, G. (2002), Improving smart card security using self-timed circuits. In *Proceedings of the Eighth International Symposium on Advanced Research in Asynchronous Circuits and Systems*.

Openswan, (2003), Openswan Project, Retrieved on May 5, 2011, from, http://www.openswan.org/.

Paul, I. (2009). *New Worm Targets Home Routers*. In *PC World*. Retrieved April 19, 2010, from http://www.pcworld.com/article/161941/nasty_new_worm_targets_home_routers_cable_modems.html.

Potlapally, N. R., Ravi, S., Raghunathan,, A., Jha, N.K. (2003), Analyzing the energy Consumption of security protocols, In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 30–35.

QuickSec IKEv2 (2009), SafeNet, Retrieved on May 23, 2011, from http://www.safenet-inc.com

Quisquater, J. J., Samyde, D. (2001), ElectroMagnetic analysis (EMA): measures and countermeasures for smart cards. In *Proceedings of the International Conference on Research in Smart Cards*, E-Smart 2001, Lecture Notes in Computer Science, (2140) 200–210.

Raghunathan, V., Schurgers, C., Park, S., Srivastava, M. (2002), Energy aware wireless microsensor networks. IEEE Signal Processing Magazine, (19), 40–50.

Raghunathan, A., Ravi, S., Hattangady, S., Quisquater, J. (2003), Securing mobile appliances: new challenges for the system designer, In *Design, Automation and Test in Europe Conference and Exhibition* (DATE'03), IEEE, 10176.

Ravi, S., Raghunathan, A., Potlapally, N. (2002), Securing wireless data: system architecture challenges, In *Proceedings of the 15th International Symposium on System Synthesis*, 195–200.

Ravi, S., Raghunathan, Kocher, P., A., Hattangady, S., (2004), Security in embedded systems: Design challenges, *Transactions on Embedded Computing Systems*, (3), 461–491.

Sakurai, K., Takagi, T. (2002), A reject timing attack on an IND-CCA2 public-key cryptosystem. In *Proceedings of ICISC 2002*, Lecture Notes in Computer Science, (2587), 359-373.

Savvides, A., Park, S., Srivastava, M. B. (2001), On modeling networks of wireless microsensors, In *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 318–319.

Shamir, A. (1999), Method and apparatus for protecting public key schemes from timing and fault attacks. *United States Patent and Trademark Office*. US Patent No. 5,991,415.

Stallings, W. (2011), Cryptography and Network Security (ed.) (13-14, 486-493). U.S.A: Pearson Education.

Stapko, T. (2008), Practical Embedded Security (62, 150-165, 180), U.S.A: Newnes (Elsevier).

Strongswan, (2004), Strongswan Project, Retrieved on May 5, 2011, from, http://www.strongswan.org/.

Sunil, D., Shubhnandan S., Jamwal D. (2010), Object Oriented versus Procedural Programming in Embedded Systems, in International Journal of Computer Science and Communication, (1-2), 47-50.

Touretzky, D.S. (2004), Gallery of CSS Descramblers. Retrieved on April 25, 2011, from, http://www.cs.cmu.edu/~dst/DeCSS/Gallery/.

William H., Kohno, T. (2010), Improving the Security and Privacy of Implantable Medical Devices, New England Journal of Medicine, (362), 1164-1166.

Wireshark (2011), Wireshark Network Traffic Analyzer Tool, Retrieved on May 5, 2011, from, http://www.wireshark.org.

Wolf, W. (Ed.) (2000), Computers as Components, In *Principles of Embedded Computing Systems Design*. Amsterdam: Elsevier.

Wouters, P., Bantoft, K. (2006), Building and Integrating Virtual Private Networks with Openswan (5-7, 9-11), Birmingham: Packt Publishing.

Wu, L., Weaver, C., Austin, T. (2001), CryptoManiac: a fast flexible architecture for secure communication, In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, 110–119.

Zurawski, R. (2006) (Ed), Embedded Systems Handbook (1.7- 1.13, 17.7- 17.13, 29.1- 29.2), U.K.: Taylor and Francis Group.