NEAR EAST UNIVERSITY

INSTITUTE OF APPLIED AND SOCIAL SCIENCES

DESIGN SIMPLE CRYPTOGRAPHY ALGORITHMS FOR INTERNET BASED EMBEDDED MICROCONTROLLER

RAED FOUAD MOSLEH ZOUROB

Master Thesis

Department of Computer Engineering

Nicosia - 2003

NEU **JURY REPORT**

DEPARTMENT OF COMPUTER ENGINEERING Academic Year: 2002-2003

STUDENT INFORMATION

Full Name	Raed Fouad Mosleh Zourob		
Undergraduate degree	**BSc.Eng.**	**Date Received**	22 Novmber 1999
Institution	Islamic Institute of Technology	**CGPA**	2.24

THESIS

Title	Design Simple Cryptography Algorithms For Internet Based Embedded Microcontroller.
Description The aim of this thesis is to provide analysis of the embedded microcontroller based systems, and to suggest suitable encryption algorithms for such systems. One of the basic problems in this field is that the resources are extremely limited and the main requirement is that the algorithm should occupy as little memory as possible, and also it should not demand large processing powers.	

Supervisor	Assoc. Prof. Dr. Doğan İbrahim	**Department**	Computer Engineering

JURY'S DECISION

The jury has decided to accept / reject the student's thesis.
The decision was taken unanimously / by majority.

JURY MEMBERS

Number Attending	3	**Date**	29/07/2003

Name	**Signature**
Prof. Dr. Fakhreddin Mamedov, Chairman of the jury	
Assoc. Prof. Dr. Rza Bashirov , Member	
Assoc. Prof. Dr.Ilham Huseynov , Member	

APPROVALS

Date
29/07/2003

Chairman of Department
Assoc. Prof. Dr. Doğan İbrahim

DEPARTMENT OF COMPUTER ENGINEERING
DEPARTMENTAL DECISION

Date: 29/07/2003

Subject: Completion of M.Sc. Thesis

Participants: Prof. Dr. Fakhreddin Mamedov, Assoc. Prof. Dr. Rza Bashirov, Assoc. Prof. Dr. Ilham Huseynov, Hani Albreem, Raid Haj Ahmed, Kamil Dimililer, Atif Munir, Abd Alhay Nabhan, Rami Matar, Mohmmed Al Hamss, Khaled Almasri, Reyad Bader.

DECISION

We certify that the student whose number and name are given below, has fulfilled all the requirements for a M .S. degree in Computer Engineering.

		CGPA
200211244	Raed Fouad Mosleh Zourob	3.714

Prof. Dr. Fakhreddin Mamedov, Committee Chairman, Dean of Engineering Faculty, NEU

Assoc. Prof. Dr. Rza Bashirov, Committee Member, Applied Math. And Computer Department, EMU

Assoc. Prof. Dr. Ilham Huseynov, Committee Member, CIS Department, NEU

Assoc. Prof. Dr. Doğan İbrahim, Supervisor, Chairman of Compute Engineering Department, NEU

Chairman of Compute Engineering Department
Assoc. Prof. Dr. Doğan İbrahim

Dedicated to my Mom and brothers

ACKNOWLEDGEMENTS

Special appreciation and gratitude to my supervisor Assoc. Prof. Dr. Doğan Akay, Department of Computer Engineering, Near East University, for his precious advice, intimate guidance and belief on my work and constant supervision throughout the MSc. Degree.

I am also thankful to Prof. Dr. Fakhreddin Mamedov, Assoc, Prof. Dr Adnan Khashman and Assoc. Prof. Dr Rahib Abiyev for their support during my studies at Near East University.

I would like to thank examining committee chairman of the jury, members and participants for there participate of this thesis.

I would like to thank my family for their constant encouragement support of my mother, brothers and sister during the preparation of this thesis

I would like to express my gratitude to Near East University for the scholarship that made the work possible.

Finally, I would also like to thank all my friends for their advice and support.

ABSTRACT

The development, over the last two years, of embedded web appliances can be considered the logical outcome of certain, different trends of progress, manufacturers concurrently began exploring possibilities in the embedded domain which could provide needed services often highly customized and requiring but a small fraction of investment on part of the customer. It is of no little consequence that the increase in capabilities of microprocessors fuelled the development in this field.

The term microcomputer is used to describe a system that includes a minimum of a microprocessor, program memory, data memory, and input/output.

A microcontroller is basically a single-chip microcomputer with additional computer such as timers, counters, analogue-to-digital counters and so on. An embedded microcontroller is a microcontroller built inside an applications system, such as a microwave oven, fridge, heavy equipment, cars etc.

The market of the web based embedded microcontroller applications is growing rapidly.

The main elective of this thesis is to investigate various security issues of embedded microcontroller and devise an algorithm for secure transfer of data to an embedded microcontroller system.

For this purpose, level low-cost, small size encryption algorithms have been studied and suitable algorithms have been proposed in the thesis.

In addition to clearly understanding the anatomy of an internet appliance, we take a brief look at the ways in which the appliance itself can affect the choice of internet protocol and how the protocols affect, in turn, the appliance. A brief treatment of the role of internet appliances in today's fast paced developments.

LIST OF GLOSSARY

ARP Address Resolution Protocol: The TCP/IP protocol that translates an Internet address into the hardware address of the network interface hardware.

Client A program that requests services from a server.

Client/server A style of computing that allows work to be distributed across hosts.

DNS Domain Name System: The name/address resolution service that uses a distributed database containing address. DNS makes it easier to refer to computers by name rather than numeric address (www.microchip.com -- instead of 198.175.253.68)

FTP File Transfer Protocol: A TCP/IP application, service, and protocol for copying files from one computer to another.

HTML Hypertext Markup Language: The language used to write pages for the Internet.

HTTP Hypertext Transfer Protocol: The TCP/IP protocol for transferring pages across the Internet.

ICMP Internet Control Message Protocol: The TCP/IP protocol used to report network errors and to determine whether a computer is available on the network.

Internet The international collection of internets that use TCP/IP to work together as one immense logical network.

IP One of the two main parts of the TCP/IP protocol suite. IP delivers TCP and UDP packets across a network.

IP Address A 32-bit unique numeric address used by a computer on a TCP/IP network.

LCP Link Control Protocol: The protocol that negotiates the parameters used by the link between two computers and is protocol within PPP.

NCP Network Control Protocol: The protocol within PPP that negotiates the type of network connection made by the two computers.

POP3 Post Office Protocol version 3: The protocol that you use to download e-mail from a POP3 mail server to your computer.

Port A number used by TCP and UDP to indicate which application is sending or receiving data.

PPP Point-to-Point Protocol: A protocol that provides serial line connectivity (that is, a dial-up with a modem) between two computers, between a computer and a network, or between two networks. PPP can handle several protocols simultaneously.

Protocol Rules and message formats for communication between computers in a network.

Protocol layers The divisions of a hierarchical network model. Each layer performs a service on behalf of the layer directly above it. Each layer receives services from the layer directly below it.

Protocol stacks A group of protocols that work together across network layers.

Server A computer program that provides services to clients, and/or the computer that runs the server program.

SMTP Simple Mail Transfer Protocol: The TCP/IP protocol for sending and receiving e-mail across a network.

Socket A data structure that allows programs on an internet to communicate. It works as a pipeline between the communicating programs and consists of an address and a port number.

TCP Transmission Control Protocol: One of the two principal components of a TCP/IP protocol suite. TCP puts data into packets and provides reliable packet delivery across a network (packets arrive in order and are not lost).

UDP User Datagram Protocol: A TCP/IP protocol found at the network (internet) layer, along with the TCP protocol. UDP sends data down to the internet layer and to the IP protocol. Unlike TCP, UDP does not guarantee reliable, sequenced packet delivery. If

data does not reach its destination, UDP does not retransmit as TCP does. Most definitions provided by the book TCP/IP for Dummies 3rd Edition by Candace Lei den and Marshall Wilensky and published by IDG Books Worldwide, Inc. ISBN 0-7645-0473-8

EEPROM*Electrically Erasable, Programmable Read-Only Memory.* (Pronounced "Double-E"-PROM.) A type of ROM that can be erased electronically

EPROM*Erasable, Programmable Read-Only Memory.* A type of ROM that can be erased by exposing it to ultraviolet light. Once erased, an EPROM can be reprogrammed with a device programmer.

Embedded system. A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In some cases, embedded systems are part of a larger system or product, as is the case of an anti-lock braking system in a car. Contrast with general-purpose computer.

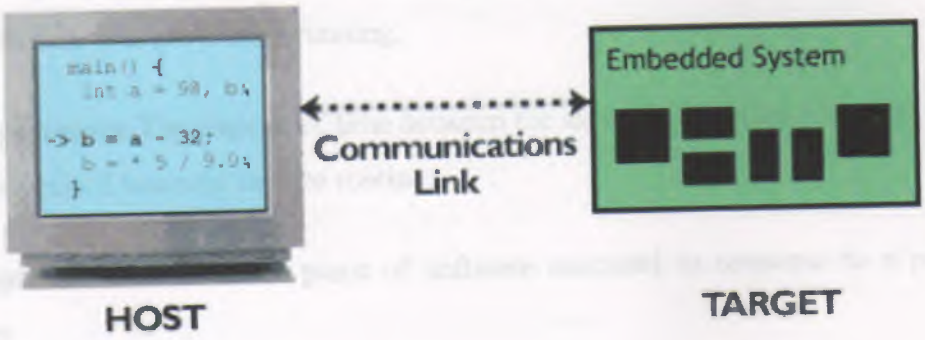
FPGA *Field Programmable Gate Array.* A type of logic chip, with thousands of internal gates, that can be programmed. FPGAs are especially popular for prototyping integrated circuit designs. However, once the design is finalized, hard-wired chips called ASICs are often used instead for their faster performance and lower cost.

Firmware Embedded software that is stored as object code within a ROM. This name is more common among the users of digital signal processors.

flash memory A RAM-ROM hybrid that can be erased and rewritten under software control. Such devices are divided into blocks, called sectors, that are individually-erasable. Flash memory is common in systems that require nonvolatile data storage at very low cost. In some cases, a large flash memory may even be used instead of a disk-drive.

high-level language A language, such as C/C++, Ada, or Java, that is processor-independent. When programming in a high-level language, it is possible to concentrate on algorithms and applications without worrying about the details of a particular processor.

Host A general-purpose computer that communicates with the target via a serial port or network connection. This term is usually used to distinguish the computer on which the debugger is running from the embedded system that is being developed.



I2C
Inter-Integrated Circuit. (Pronounced "Eye Squared C.") An I2C bus is an inexpensive type of chip interconnection that is popular on circuit boards. Compare with 1-Wire and SPI.

ICE *In-Circuit Emulator*.

I/O *Input/Output*. The interface between a processor and the world around it. The simplest examples are switches (inputs) and LEDs (outputs).

I/O device A piece of hardware that interfaces between the processor and the outside world. Common examples are switches and LEDs, serial ports, and network controllers. Also called a peripheral.

I/O map A table or diagram containing the name and address range of each I/O device addressable by the processor within the I/O space. I/O maps are a helpful aid in getting to know the target.

I/O space A special memory region provided by some processors and generally reserved for the attachment of I/O devices. Memory locations and registers within an I/O space can be accessed only via special instructions. For example, processors in the 80x86 family have special I/O space instructions called in and out. Contrast with memory space.

Interrupt An asynchronous electrical signal from a peripheral to the processor. When the peripheral asserts this signal, we say that an interrupt occurs. When an interrupt occurs, the current context is saved and an interrupt service routine is executed. When the interrupt service routine exits, control of the processor is returned to whatever part of the software was previously running.

interrupt latency The amount of time between the assertion of an interrupt and the start of the associated interrupt service routine.

interrupt service routine A piece of software executed in response to a particular interrupt.

interrupt type A unique number associated with each interrupt.

interrupt vector The address of an interrupt service routine.

interrupt vector table A table containing interrupt vectors and indexed by interrupt type. This table contains the processor's mapping between interrupts and interrupt service routines and must be initialized before the first interrupt occurs.

intertask communication/synchronization A mechanism used by tasks and interrupt service routines to share information or synchronize their activities or access to shared resources. The most common building blocks of intertask communication/synchronization are mutexes and semaphores. However, some operating systems support more advanced mechanisms such as message queues or monitors.

Linker A software development tool that accepts one or more object files as input and outputs a relocatable program. The linker is thus run after all of the source files have been compiled or assembled.

memory map A table or diagram containing the name and address range of each peripheral addressable by the processor within the memory space. Memory maps are a helpful aid in getting to know the target.

memory-mapped I/O A common hardware design methodology in which I/O devices are placed into the memory space rather than the I/O space. From the processor's point of view, memory-mapped I/O devices look very much like memory devices.

memory space A processor's standard address space. Contrast with I/O space.

Microcontroller A microcontroller is very similar to a microprocessor. The main difference is that a microcontroller is designed specifically for use in embedded systems. Microcontrollers typically include a CPU, memory (a small amount of RAM and/or ROM), and other peripherals on the same chip. Common examples are the PIC and 8051, Intel's 80196, and Motorola's 68HCxx series.

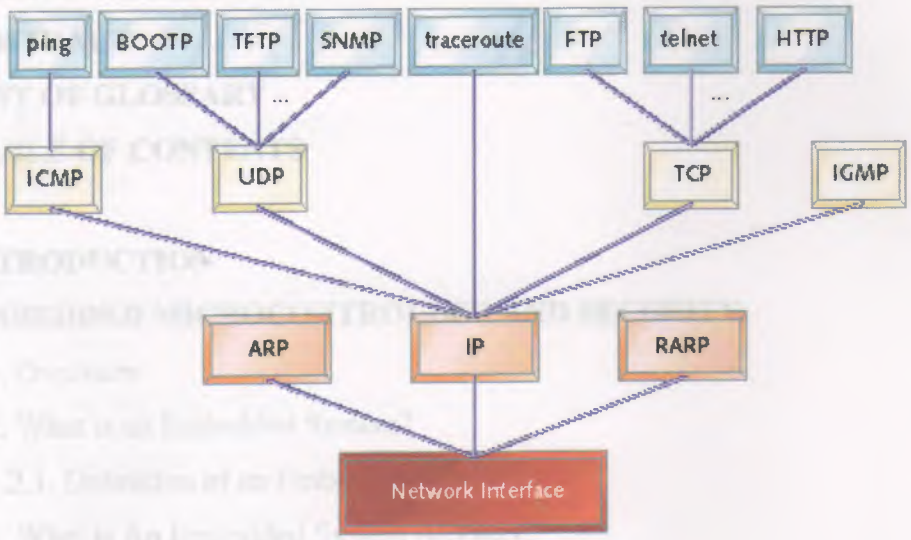
Microprocessor A piece of silicon containing a general-purpose CPU. The most common examples are Intel's 80x86 and Motorola's 680x0 families

Opcode A sequence of bits that is recognized by the processor as one of the instructions in its instruction set.

PROM *Programmable Read-Only Memory*. A type of ROM that can be written (programmed) with a device programmer. These memory devices can be programmed only once, so they are sometimes referred to as write-once or one-time programmable.

protocol stack Any set of communication protocols, such as TCP/IP, that consists of two or more layers of software and hardware. It's called a stack because each layer builds on the functionality in the layer below. For example, in TCP/IP parlance, the lowest layer is called the *physical layer*. That's where the rubber meets the road; or, more accurately, the bits meet the communications medium at the network interface. Above that is the *data link layer*, which gives each device on the network its unique address. These first two layers of the TCP/IP protocol stack are typically implemented in hardware. Once the networked devices have addresses, they can communicate. That's where layer three, the *network layer*, comes in. IP is just one of the protocols that exists at this level in a TCP/IP stack; TCP and UDP are competing protocols at the *transport layer*. Three more layers of software (*session*, *presentation*, and *application*) are defined above those, thus completing the 7-layer *OSI Reference Model*. Only layers 1-2 (hardware) and 3-5 (software) are shown in this figure; some individual protocols are

excluded. When data is sent across the network, it generally begins at layer 5 or above, travels down through the protocol stack on the sending system, out onto the network, then back up the stack on the receiving system.



Pulse width modulation A technique for controlling analog circuits with a processor's digital outputs. PWM is employed in a wide range of applications, from measurement and communications to power control and conversion.

Target Another name for the embedded system. This term is usually used during software development, to distinguish the embedded system from the host with which it communicates.

TABLE OF CONTENTS

INTRODUCTION

Microcontrollers are general purpose microprocessors which have additional parts that allow them to control external devices. Basically, a microcontroller excites a user program which is loaded in its program memory. Vader the control of this program, data is received from external devices, manipulated and then sent to external output devices.

Embedded systems are designed to reform specific, usually dedicated tasks. These systems are designed using microcontroller as their processing elements. Such systems have very limited resources, usually very small program and data memories, and low processing speed; the major advantage of microcontroller systems is their externally low cost.

The aim of this thesis is to provide analysis of the embedded microcontroller based systems, and to suggest suitable encryption algorithms for such systems. One of the basic problems in this field is that the resources are extremely limited and the main requirement is that the algorithm should occupy as little memory as possible, and also it should not demand large processing powers.

The thesis consists of the introduction and five chapters.

Chapter 1 describes the basic principles of embedded systems and their security requirement, brief overview on viruses and related threats.

Chapter 2 present the commonly used internet protocols and also describes how the web appliances can be designed using these protocols. IP Addressing and message/packet sending criteria is defined.

Chapter 3 studies the basic internet security issues with respect to microcontrollers and presents some of the techniques used to enhance internet security and available cryptographic engines.

Chapter 4 the principles of microcontroller systems and their interfacing techniques are decided.

Chapter 5 is the main part if the thesis and this chapter describe various small- size encryption algorithms which are suitable to embedded controller. Programs are developed by the author to simulate carious algorithms in order to access their suitability to low-cost microcontroller based security applications.

Finally conclusion section presents the results of the study and values recommendations for future work in this field.

1. EMBEDDED MICROCONTROLLER AND SECURITY THREATS

1.1 Overview

The growth in connectivity with regards to speed of connection and number of people connecting to the Internet has spawned great many web-related services. Our focus is principally on microcontroller based devices able to connect to the Internet without using a PC.

The chip we have chosen is iChip S7600-A and the iReady tuner technology which offers more efficient and less cost solutions using TCP/IP stack, it is a completely self-contained, hardware drop-in solution providing dial-up connectivity to the Internet. We have chosen the Seiko Instruments iChip because it is ideal for any application for uploading files over the Internet through secure means in an automatic mode, having the critical communications functions in pure hardware means extremely high reliability as there is no risk of a software 'crash' as is seen in so many PC software programs. The controller can support virtually any type of input ballot medium, such as punch card, touch screen, or optical mark sense, just by inserting a different PCMCIA card which contains the appropriate applications software. It requires no monitor or keyboard. Election officials simply press one button on the device to scan ballot information and another button to send the collected results, via a modem, to a central location.

Traditionally the PC was considered the principal device that could connect to the Internet. Only recently have developments made it possible to build microcontroller-based systems that can negotiate a connection to the Internet. These developments are listed as:

The convergence of network standards. Before the acceptance of TCP/IP as an Internet standard, manufacturers developed proprietary, largely incompatible standards, making it difficult to generate services uniformly to vast number of users.

The availability of custom made ASICS as well as off the shelf chips incorporating both a CPU as well as network components.

The emergence of software tools which help designers with the task of packing their product with more features.

This thesis focuses on the various aspects of Web Appliances, their anatomy, purpose and relevance.

1.2. What is an Embedded System?

This seems like an almost impossible question to answer since the topic is so vast. The thing is, *embedded systems* are all around us. They can be found in cars, watches, mobile phones, stereos, hand-held devices, as well as elevators, just to name a few. Nowadays, almost everything electric-powered contain some sort of *embedded Intelligence*, you can say that all these everyday things get their "smarts" from embedded systems [22].

An *embedded system* consists of computer hard wares and soft wares that in turn form a component of a larger system that is expected to function without human interference.

Embedded Systems are designed to perform a specific, dedicated function. They typically have tight constraints on functionality and implementation. They also must guarantee *real-time operation (RTOS)* reactive to external events, conform to size and weight limits, budget power and cooling consumption, satisfy safety and reliability requirements, and meet cost targets as well as time constraints[4].

An example of *embedded systems* can be found in your car's *antilock braking system*. It uses a *real-time operating system (RTOS)* acting as its *"brain"*. *A RTOS* combines predictable response times and behaviors and can coordinate functions without missing a beat.

So in other words, a *real-time operating system (RTOS)* will guarantee speedy delivery of certain functions within an appropriate response time[6].

In addition, *embedded systems* consist of a single microprocessor board with the software stored in *ROM. ROM* stands for *Read-Only Memory*, which is a small memory allowing fast access to permanently-stored data, yet prevents addition to or modification of the data. The software in the system may perform anything from data logging

functions to advanced computations. An *embedded system* may even include some sort of operating system. In recent years, *Linux* has become a popular choice to be embedded into many applications. Oftentimes an embedded system is simple enough to be written as a single program.

Embedded systems employ software that is usually written in high-level languages such as "C" and is targeted to run on an "embedded" boards or systems. They are also designed to be invisible to users, thus although they are everywhere, many people have no idea what they are [4].

1.2.1. Definition of an Embedded System:

An "embedded system" is any computer system or computing device that performs a dedicated function or is designed for use with a specific embedded software application. Embedded systems may use a ROM-based operating system or they may use a disk-based system, like a PC. But an embedded system cannot be used as a commercially viable substitute for multipurpose computers or devices [37].

1.3. What is An Embedded System Anyway?

Typically embedded systems are microcomputer systems with software self-contained in Read-Only Memory (ROM) and without a readily-recognizable software operating system. In essence, the programmer has written a specialized operating system in the process of making the hardware function. Embedded systems are most often "dedicated" microprocessor environments with single functional utilization. Frequently, the systems are "black boxes" associated with manufacturing equipment, process control devices, or similar industrial hard ware. These boxes generally provide the common benefits of directing, controlling, or monitoring specific functions of specific devices. They all contain microcomputer systems with software that is totally ROM-based, and they usually have programmable interfaces. The capabilities of individual systems, the devices attached, and the methods of gathering information from the operator and of disseminating information to the operator (operator/interface), are ostensibly unique, but have similar conceptual and technical characteristics [4].

With these common traits revealed, there is still no way to precisely define those systems that fall into the embedded category. A comprehensive and concise definition

of embedded systems probably does not exist. Even the United States Department of Defense does not attempt definition within its series of "Military Standard" publications. If the operating system were removed from a typical personal computer (PG), and all the programs that were to be executed in a given environment were made to reside in ROM, then the modified PG might be considered an embedded system with processing dedicated to predetermined functions, but this is a stretch.

Better examples are found inside consumer products. Many American homes have microwave ovens, dishwashers, security systems, audiovisual entertainment centers, or communications equipment that use programmable embedded systems to monitor or control these units. The owner can "program" features of the device via touch panels buttons, or hand-held remote controllers to accomplish a desired end and the system does the rest. Some controlled operations are sequential, some are timed, and some are one-shot commands to the equipment.

In this case, it appears that minimal cost and optimum design could be achieved with a PC and a versatile, pre-emptive, multitasking operating system. In addition, the PC approach would provide the requisite interoperability and flexibility to accommodate the fast-paced evolution of computer technologies, as well as the slower, but ultimately ensuing, evolution of customer requirements. Considerations of this nature may prove to be a real concern for manufacturers of specialized equipment in many industrial areas.

1.4. A Typical Embedded System

Embedded systems are used in a vast array of applications across many industries; however, most that will be encountered by consultants and others outside a specific market environment will be of a control nature for plant floor equipment or process control. Manufacturers of less complex, small systems seldom ask for help outside their organizations. Those who deal in larger and more complex systems frequently have other engineering focus, greater workloads, and a myriad of other reasons to ask for occasional outside help.

Technically, there are prevalent and common characteristics of embedded systems. From a programmer's perspective the following components are a minimum: Central Processing Unit (CPU), Random Access Memory (RAM), Programmable Read Only

Memory (PROM) or Erasable PROM (EPROM), and Input/Output (I/O) space [4].

Since many embedded systems contain time-critical functions, another of ten essential components is one or more system timers. Timer units are available in a variety of commercial forms. For boxes built around the VME or STD system bus architectures, timers are frequently included as features of CPU boards manufactured for these systems, or as separate specialized timer boards." CPUs sometimes provide timers for programming purposes; Intel's 80186, for example, have three.

Device interaction is the sole reason for the system's existence, so application-specific devices are also ultimate system essentials. One or more devices will be used for operator interaction with the system (operator interface). All devices will be attached to the system via the I/O space. From the programmer's perspective, a baseline embedded system is shown in Figure 4.2

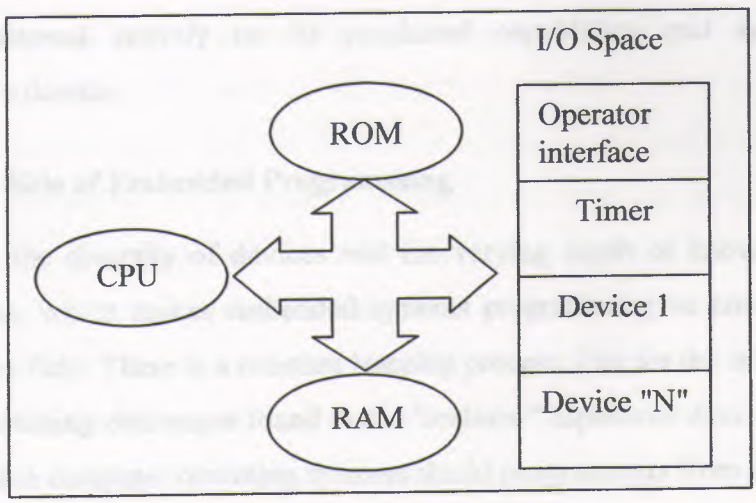


Figure 4.2 Baseline Embedded System

Obviously, the devices above can vary drastically from environment to environment. For the spray equipment mentioned earlier, there might be multiple heaters, pumps, indicator lights, and interlocks as a minimum. A serial communications interface could allow remote configuration, and certainly a display and keypad would be required. Feedback from a conveyer could allow automatic adjustment of pump pressure for varying production-line speeds. For a carpet metering and cutting machine, there could be multiple photo eyes and conveyer motors. A measuring wheel, a cutting mechanism,

a display, and a keypad would be desirable. Several serial communications interfaces could allow centralized reporting and/or bar-coded inputs for preprinted orders.

Operator interfaces are equally disparate devices. Control boxes for large plant floor machinery sometimes use serial interfaces to giant high-resolution color screens sporting Carroll infrared touch technology. The huge physical size allows a single individual in the plant to monitor several machines from a comfortable distance. Stand-alone embedded units typically have small keypads and small liquid crystal diode (LCD) displays that show only a few lines of text, or only a single text line of a few characters. This allows more economical construction when feasible. For further economy under the right circumstances, interfaces may even be digital switches for operator input with only indicator lights for problem determination.

Regardless of the devices used in the system, the programmer will be required to write some level of driver for each device. The depth of knowledge required for a single device will depend entirely on its purchased capabilities and system specific implementation details.

1.4.1. The Up Side of Embedded Programming

It is precisely the diversity of devices and the varying depth of knowledge required about each one, which makes embedded systems programming so attractive to those who choose the field. There is a constant learning process. Fun for the initiated includes the substantial timing challenges found in the "realtime" aspects of device coordination. In addition, since computer operating systems shield programmers from many low-level functions of hardware and peripheral I/O, the programming of embedded systems requires more in-depth knowledge of how platform components function and interact than does programming under an operating system.

Embedded systems programming is much closer to the hardware. In actual practice, restraints of economy can affect the quality and capability of selected components for a given system. When organizations must require that specific system board costs do not exceed a given figure, substantial programming challenges that translate to "drudgery" sometimes ensue as with anything there are good and bad projects.

The makeup of excellent programmers is still an elusive compound. Some are graduate musicians and airplane pilots rather than computer engineering majors. Whatever the ingredients, they share the common trait of project dedication born of genuine interest. All enjoy savoring the fruits of their labor when, for example, a control unit does exactly what it is supposed to do. A casual observer looking at computer-controlled machinery would see only moving parts; the programmer sees the coordinated execution of each code thread performing its function. Sometimes this is reward enough; but for those programmers with a less technical bent, embedded systems programming can be a real pain. Some things are better left unknown, and, to many, this includes device details and other aspects of the field.

1.5. Embedded Web Server VS. Client

When most people think of the Internet, they think about viewing web pages filled with information. A computer of some sort provides the necessary resources required to support a web server. The computer has significant hard disk space to hold the web pages and ancillary data files such as data sheets, application notes, etc. It also has a high-speed communications interface such as a T1 line, a 56K modem, etc., that can serve this information to a user quickly. Embedded systems are quite different from standard web servers. They have limited resources in terms of memory space and operating speed. Embedded systems usually work with a few kilobytes, up to several megabytes of memory and operate up to 40 MHz. Typical PCs have several gigabytes of memory storage and are starting to operate in the gigahertz range. Therefore, web servers are not really practical for embedded applications, but the embedded web client is very practical. One of the most viable embedded web client applications is a vending machine.

The microcontroller keeps track of all functions:

- Price for each item.
- Number of items left.
- Amount of deposited money.
- Amount of money available for change.

One could also argue that the machine could keep track of the name or type of items that it is dispensing. On a preset interval (such as once a night between 11 PM and 4

AM), the vending machine would dial out and make a connection to a local Internet Service Provider (ISP). The machine would then connect to a server and upload all the information. If the machine is out of change, full of deposited money, or needs restocking, a report can be generated detailing exactly what is required. The service person can now make one trip to the machine because they know exactly what that particular machine needs. The machine can also call out if an operating error has been detected. This may include the vending machine is tilted to improperly dispense items, or lights have burned out. A service report is generated and the service person knows exactly what is needed to fix the problems. Since we have established a full duplex link, the embedded Internet connection can provide additional benefits, such as reprogramming the microcontroller with new firmware to fix bugs, or new pricing structure for items.

1.6. Security Incidents

A *security incident* is any network-related activity with negative security implications. This usually means that the activity violates an explicit or implicit security policy (see the section on security policy). Incidents come in all shapes and sizes. They can come from anywhere on the Internet, although some attacks must be launched from specific systems or networks and some require access to special accounts. An intrusion may be a comparatively minor event involving a single site or a major event in which tens of thousands of sites are compromised. (When reading accounts of incidents, it is observed that different groups may use different criterion, for determining the bounds of an incident.). A typical attack pattern consists of gaining access to a user's account, gaining privileged access, and using the victim's system as a launch platform for attacks on other sites. It is possible to accomplish all these steps manually in as little as 45 seconds; with automation, the time decreases further.

1.7. Sources of Incidents

It is difficult to characterize the people who cause incidents but I have collected the information from different location. like "Mr. James Niccolai" April 04, 2003 said that "The number of computer security incidents and attacks detected at businesses worldwide soared by 37 percent between the fourth quarter of 2002 and the first quarter

of this year, fueled in part by a surge in the number of mass-mailing worms", according to a report due out Monday from Internet Security Systems Inc [46].

"1123 incidents were reported to AusCERT by the end of February 2000, compared with 235 incidents by the same time last year," Mr. McMillan said "The largest reported increase was in the commercial and education sectors" [47].

The number of security-related incidents reported to the Team each year since 1988 to 1998 from Computer Emergency Response Team [48]. Computer Security Incident Response Teams [49].

An intruder may be an adolescent who is curious about what he or she can do on the Internet, a college student who has created a new software tool, an individual seeking personal gain, or a paid "spy" seeking information for the economic advantage of a corporation or foreign country. An incident may also be caused by a disgruntled former employee or a consultant who gained network information while working with a company. An intruder may seek entertainment, intellectual challenge, and a sense of power, political attention, or financial gain.

Years of incidence	No of incidence
1988	6
1989	132
1990	252
1991	406
1992	773
1993	1334
1994	2340
1995	2412
1996	2573
1997	2134
1998	3734
1999	8268
2000	21756
2001	52658
2002	43136
2003	N/A

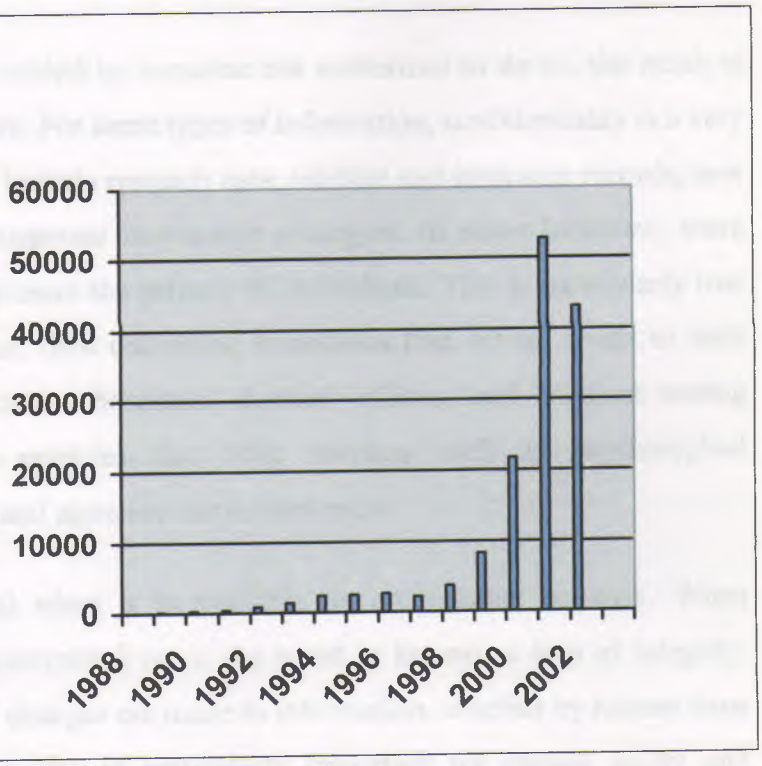


Figure 1.1 Growths in Security Incidence [10], [46], [47], [48], [49]

One characteristic of the intruder community as a whole is its communication. There are electronic newsgroups and print publications on the latest intrusion techniques, as well as conferences on the topic. Intruders identify and publicize un-configured systems; they use those systems to exchange pirated software, credit card numbers, exploitation programs, and the identity of sites that have been compromised, including account names and passwords. By sharing knowledge and easy-to-use software tools, successful intruders increase their number and their impact.

The data for 1995 and partial data for 1996 show a slowing of the rate at which incidents are reported to the CERT/CC (perhaps because of sites' increased security efforts or the significant increase in other response teams formed to handle incidents). However, the rate continues to increase for serious incidents, such as root compromises, services outages, and packet sniffers.

1.8. Basic Security Concepts

Three basic security concepts important to information on the Internet are confidentiality, integrity, and availability. Concepts relating to the people who use that information are authentication, authorization, and nonrepudiation.

When information is read or copied by someone not authorized to do so, the result is known as loss of confidentiality. For some types of information, confidentiality is a very important attribute. Examples include research data, medical and insurance records, new product specifications, and corporate investment strategies. In some locations, there may be a legal obligation to protect the privacy of individuals. This is particularly true for banks and loan companies; debt collectors; businesses that extend credit to their customers or issue credit cards; hospitals, doctors' offices, and medical testing laboratories; individuals or agencies that offer services such as psychological counseling or drug treatment; and agencies that collect taxes.

Information can be corrupted when it is available on an insecure network. When information is modified in unexpected ways, the result is known as loss of integrity. This means that unauthorized changes are made to information, whether by human error or intentional tampering. Integrity is particularly important for critical safety and

financial data used for activities such as electronic funds transfers, air traffic control, and financial accounting [8].

1.9. Why to Care about Security?

It is remarkably easy to gain unauthorized access to information in an insecure networked environment, and it is hard to catch the intruders. Even if users have nothing stored on their computer that they consider important, that computer can be a "weak link", allowing unauthorized access to the organization's systems and information.

Seemingly innocuous information can expose a computer system to compromise. Information that intruders find useful includes which hardware and software are being used, system configuration, type of network connections, phone numbers, and access and authentication procedures. Security-related information can enable unauthorized individuals to get access to important files and programs, thus compromising the security of the system. Examples of important information are passwords, access control files and keys, personnel information, and encryption algorithms.

The consequences of a break-in cover a broad range of possibilities: a minor loss of time in recovering from the problem, a decrease in productivity, a significant loss of money or staff-hours, a devastating loss of credibility or market opportunity, a business no longer able to compete, legal liability, and the loss of life.

1.10. Cryptography

One of the primary reasons that intruders can be successful is that most of the information they acquire from a system is in a form that they can read and comprehend. When you consider the millions of electronic messages that traverse the Internet each day, it is easy to see how a well placed network sniffer might capture a wealth of information that users would not like to have disclosed to unintended readers. Intruders may reveal the information to others, modify it to misrepresent an individual or organization, or use it to launch an attack. One solution to this problem is, through the use of cryptography, to prevent intruders from being able to use the information that they capture[10].

Encryption is the process of translating information from its original form (called *plaintext*) into an encoded, incomprehensible form (called *ciphertext*). Decryption refers to the process of taking ciphertext and translating it back into plaintext. Any type of data may be encrypted, including digitized images and sounds.

Cryptography secures information by protecting its confidentiality. Cryptography can also be used to protect information about the integrity and authenticity of data. For example, checksums are often used to verify the integrity of a block of information. A checksum, which is a number calculated from the contents of a file, can be used to determine if the contents are correct. An intruder, however, may be able to forge the checksum after modifying the block of information. Unless the checksum is protected, such modification might not be detected. Cryptographic checksums (also called message digests) help prevent undetected modification of information by encrypting the checksum in a way that makes the checksum unique.

The authenticity of data can be protected in a similar way. For example, to transmit information to a colleague by Email, the sender first encrypts the information to protect its confidentiality and then attaches an encrypted digital signature to the message. When the colleague receives the message, he or she checks the origin of the message by using a key to verify the sender's digital signature and decrypts the information using the corresponding decryption key. To protect against the chance of intruders modifying or forging the information in transit, digital signatures are formed by encrypting a combination of a checksum of the information and the author's unique private key. A side effect of such authentication is the concept of nonrepudiation. A person who places their cryptographic digital signature on an electronic document cannot later claim that they did not sign it, since in theory they are the only one who could have created the correct signature.

Current laws in several countries, including the United States, restrict cryptographic technology from export or import across national borders. In the era of the Internet, it is particularly important to be aware of all applicable local and foreign regulations governing the use of cryptography.

1.10.1. Terminology

Encryption was the first cryptographic operation used to ensure secrecy or confidentiality of information transmitted across an insecure communication channel. The encryption operation takes a piece of information (also called the message, message block, or plaintext) and translates it into a cryptogram (ciphertext or codeword) using a secret cryptographic key. Decryption is the reverse operation to encryption. The receiver who holds the correct secret key can recover the message (plaintext) from the cryptogram (ciphertext).

The step-by-step description of encryption (or decryption) is called the en*cryption algorithm* (or decryption *algorithm).* If there is no need to distinguish encryption from decryption, we are going to call them collectively ciphers, *Crypto algorithm* or *cryptosystems.*

Private-key or *symmetric* cryptosystems use the same secret key for encryption and decryption. More precisely, the encryption and decryption keys do not need to be identical the knowledge of one of them suffices to find the other (both keys must be kept secret).

Pnblic-key or asymmetric cryptosystems use different keys for encryption and decryption. The knowledge of one key does not compromise the other. [8].

1.11. Web Security

Virtually all businesses, most government agencies, and many individuals now have Web sites. The number of individuals and companies Internet access is expanding rapidly, and all of these have browsers. As a result, businesses are enthusiastic about setting up facilities on the Web for electronic commerce. But the reality is that the Internet and the Web are extremely vulnerable to compromises of various sorts. As businesses wake up to this reality, the demand for secure Web services grows.

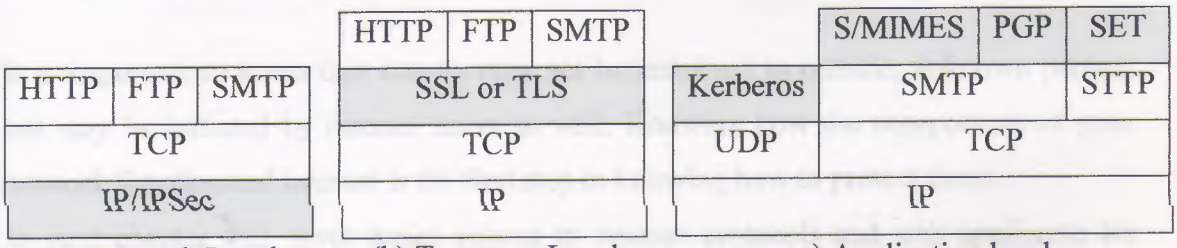
The topic of Web security is a broad one and can easily fill a book (several are recommended at the end of this chapter). In this chapter, we begin with a discussion of the general requirements for Web security and then focus on two standardized schemes that are becoming increasingly important as part of Web commerce: SSL/TLS and SET.

1.11.1. Web Traffic Security Approaches

A number of approaches to providing Web security are possible. The various approaches that have been considered are similar in the services they provide and. to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack, Figure 1.2 illustrates this difference. One way to provide Web security is I use IP Security (Figure 1.2a). The advantage of using IPSec is that it is transparent to end users and applications and provides a general-purpose solution. Further IPSec includes a filtering capability so that only selected traffic need incur the over head of IPSec processing [2].

Another relatively general-purpose solution is to implement security just above TCP (Figure 1.2). The foremost example of this approach is the security Sockets Layer (SSL) and the follow-on Internet standard of SSL knows transport Layer Security (TLS). At this level, there are two implementation Choices. For full generality, SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications. Alternatively, SSL can be embedded in specific packages. For example, Netscape and Microsoft Explorer browser come equipped with SSL, and most Web servers have implemented the protocol.

Application-specific security services are embedded within the application. Figure 1.2c shows examples of this architecture. The advantage of this



(a) Network Level (b) Transport Level c) Application level

Figure 1.2 Relative Locations of Security Facilities in the TCP-IP Protocol Stack.

Approach is that the service can be tailored to the specific needs of a given application. In the context of Web security, an important example of this approach is Secure Electronic Transaction (SET) [2].

1.12. Summary

It is obvious at the end of this chapter that growth in connectivity with regards to speed of connection and number of people connecting to the Internet has spawned great many web-related services, and how the embedded systems play a vital role in providing efficient and low cost solution.

Also we have seen in this chapter the S7600-A and the iReady tuner technology was chose because it offers more efficient and less cost solutions using TCP/IP stack and is a completely self-contained, hardware drop-in solution providing dial-up connectivity to the Internet, and not to require any monitor or keyboard. Election officials simply press one button on the device to scan ballot information and another button to send the collected results, via a modem, to a central location.

We must walk a fine line between closing as many doors as possible without encouraging trusted users to try to circumvent the policy because it is too complex and time-consuming to use.

Allowing Internet access from an organization poses the most risk to that organization. This chapter has outlined the types of attacks that may be possible without a suitable level of protection. If a compromise occurs, tools and applications are available to help flag possible vulnerabilities before they occur or to at least help the network administrator monitor the state of the network and its resources.

It is important to stress that attacks may not be restricted to outside, unknown parties, but may be initiated by internal users as well. Knowing how the components of your network function and interact is the first step to knowing how to protect them.
In next chapter will cover topics related to internet protocols and web applicants for microcontroller.

2. WEB APPLIANCES AND THE INTERNET PROTOCOLS

2.1. Overview

The Internet protocols are the world's most popular open-system (nonproprietary) protocol suite because they can be used to communicate across any set of interconnected networks and are equally well suited for LAN and WAN communications. The Internet protocols consist of a suite of communication protocols, of which the two best known are the Transmission Control Protocol (TCP) and the Internet Protocol (IP). The Internet protocol suite not only includes lower-layer protocols (such as TCP and IP), but it also specifies common applications such as electronic mail, terminal emulation, and file transfer. This chapter provides a broad introduction to specifications that comprise the Internet protocols. Discussions include IP addressing and key upper-layer protocols used in the Internet.

Documentation of the Internet protocols (including new or revised protocols) and policies are specified in technical reports called Request for Comments (RFCs), which are published and then reviewed and analyzed by the Internet community. Protocol refinements are published in the new RFCs. To illustrate the scope of the Internet protocols, Figure 2.1 maps many of the protocols of the Internet protocol suite and their corresponding OSI layers. This chapter addresses the basic elements and operations of these and other key Internet protocols.

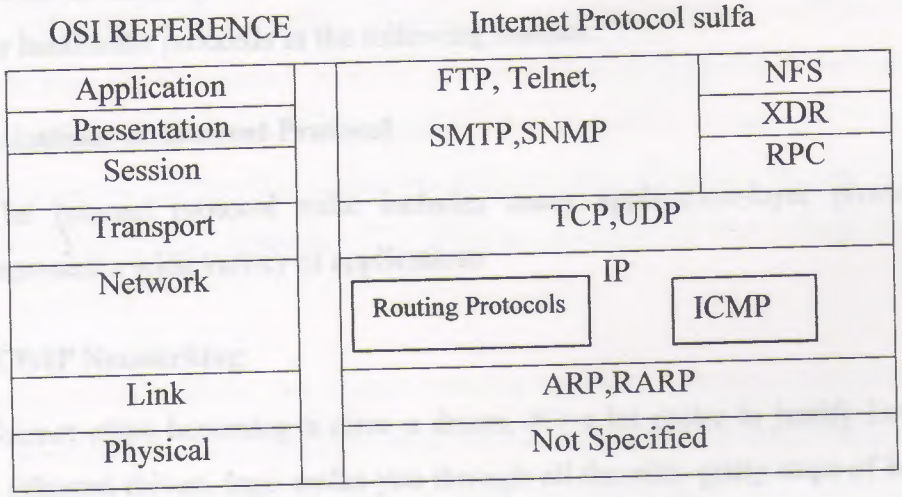


Figure 2.1 Internet protocols span the complete range of OSI model layers.

2.2. Internet Protocol (IP)

The Internet Protocol (IP) is a network-layer (Layer 3) protocol that contains addressing information and some control information that enables packets to be routed. IP is documented in RFC 791 and is the primary network-layer protocol in the Internet protocol suite. Along with the Transmission Control Protocol (TCP), IP represents the heart of the Internet protocols. IP has two primary responsibilities: providing connectionless, best-effort delivery of datagrams through an internet work; and providing fragmentation and reassembly of datagrams to support data links with different maximum-transmission unit (MTU) sizes.

2.3. TCP Connection Establishment

To use reliable transport services, TCP hosts must establish a connection-oriented session with one another. Connection establishment is performed by using a "three way handshake" mechanism.

A three-way handshake synchronizes both ends of a connection by allowing both sides to agree upon initial sequence numbers. This mechanism also guarantees that both sides are ready to transmit data and know that the other side is ready to transmit as well. This is necessary so that packets are not transmitted or retransmitted during session establishment or after session termination. Each host randomly chooses a sequence number used to track bytes within the stream it is sending and receiving. Then, the three-way handshake proceeds in the following manner:

2.4. Applications of Internet Protocol

- The Internet protocol suite includes many application-layer protocols that represent a wide variety of applications

2.4.1. TCP/IP Networking

With Ethernet chips becoming a dime a dozen, it's a lot easier to justify having your network Ethernet driven. Ingo walks you through all the nitty-gritty steps of how to get your Ethernet-based device working with real-time capabilities.

First question: *what do I define as an Ethernet-based device?*

The device has an Ethernet port, as a primary interface. A temperature sensor with an Ethernet port is one example, but I'm not sure about the economics of such a project. To make developing such a device more interesting to me, it needs to solve one of my problems. So, here goes.

I frequently use a prototyping system called *a logic engine*, which is essentially a system tester. It has 128 I/O ports and is controlled via a PC-compatible parallel port.

The device I want to discuss is an Ethernet-to-parallel port device that controls the logic engine. In a sense, the parallel port–based logic engine becomes an Ethernet-based logic engine. This technique has several advantages. First of all, I don't need to write and install a device driver on the system controlling the logic engine. Secondly, I can share the logic engine between workstations or over the Internet.

Finally, I can use a variety of programming languages on many platforms to talk to this device. All my language on the workstation needs now is a network library. I don't have to write various libraries like I did for the old parallel-based interface.
But before getting started on this thesis, I want to take a look at one of its major components-TCP/IP.

2.4.2. TCP/IP Networking Promer

Recall that TCP/IP implementations use the socket API for application programs to interface to the stack In a nutshell, when two programs want to establish a TCP/IP connection, they set up a socket which is identified with an Internet address and a port. Once the connection is set up, the programs write data to the socket as if it were a file and the data comes out the other end of the socket, where the other program reads it like a file. The protocol stack treats the TCP/IP as an unstructured byte stream, so it's easy to program for TCP/IP.

But as you probably know, the actual network interface between computers usually consists of a LAN or serial interface, with data being transferred in packets. Let's look at what goes on in the TCP/IP stack to give us this illusion of a connection. The protocol stack for TCP/IP is illustrated in Figure 2.5. At the top, we have the application

layer, which is followed by the transport and Internet work layers. The network interface and the network are at the bottom.

The application program, in the application layer, is considered part of the protocol stack. The application usually implements some kind of protocol on top of the unstructured byte stream provided by TCP/IP.

For example, Internet mail uses the Simple Mail Transport Protocol (SMTP) to exchange E-mail messages between different hosts on the Internet. You may also be familiar with the Hypertext Transport Protocol (HTTP) that Web browsers and servers use to communicate.

Note that application-layer protocols are implemented in the application and that the interface between the application layer and the transport layer is the socket API. The rest of the protocol stack the transport layer through the network interface is usually implemented in the OS because its implementation is the same for any application.

The transport layer implements the byte stream abstraction of the transmission control protocol (TCP) over the packet-switched Internet work protocol (IP). TCP implements what networking folks call a reliable virtual circuit. It is reliable because the implementation makes sure that the data, once received, is not corrupted by noise, arrives in order, and nothing is missing or duplicated. A virtual circuit is kind of like a phone connection. Once the connection is made, it is maintained until disconnected. TCP implements this reliable virtual circuit by maintaining a state on each end of the connection. There is state information for each active (or connected) port. The status information maintains the connection state (i.e., whether the circuit is being set up or torn down and whether it is connected). Sequence pointers are kept to indicate which byte of the stream has been received, acknowledged, and delivered to the receiver. The transmitter also tracks which byte it sent, how much data is left to send, and the last byte acknowledged by the receiver. Each time one side of the connection has data to send (TCP is full-duplex, by the way), it bundles up the data into a packet and adds a TCP header to the packet.

The packet header includes an end-to-end checksum to make sure that the TCP packet is received intact, a copy of the sequence pointer of the sent data, and a copy of the

sequence number for the data it last received, which serves as the acknowledgment. Including the acknowledgement for the data received, while transmitting data in the other direction is referred to as piggybacking the acknowledgment. The header also contains the destination port number this TCP packet goes to and the source port number from which it was sent. Figure 2.6 shows how the various sequence pointers are related.

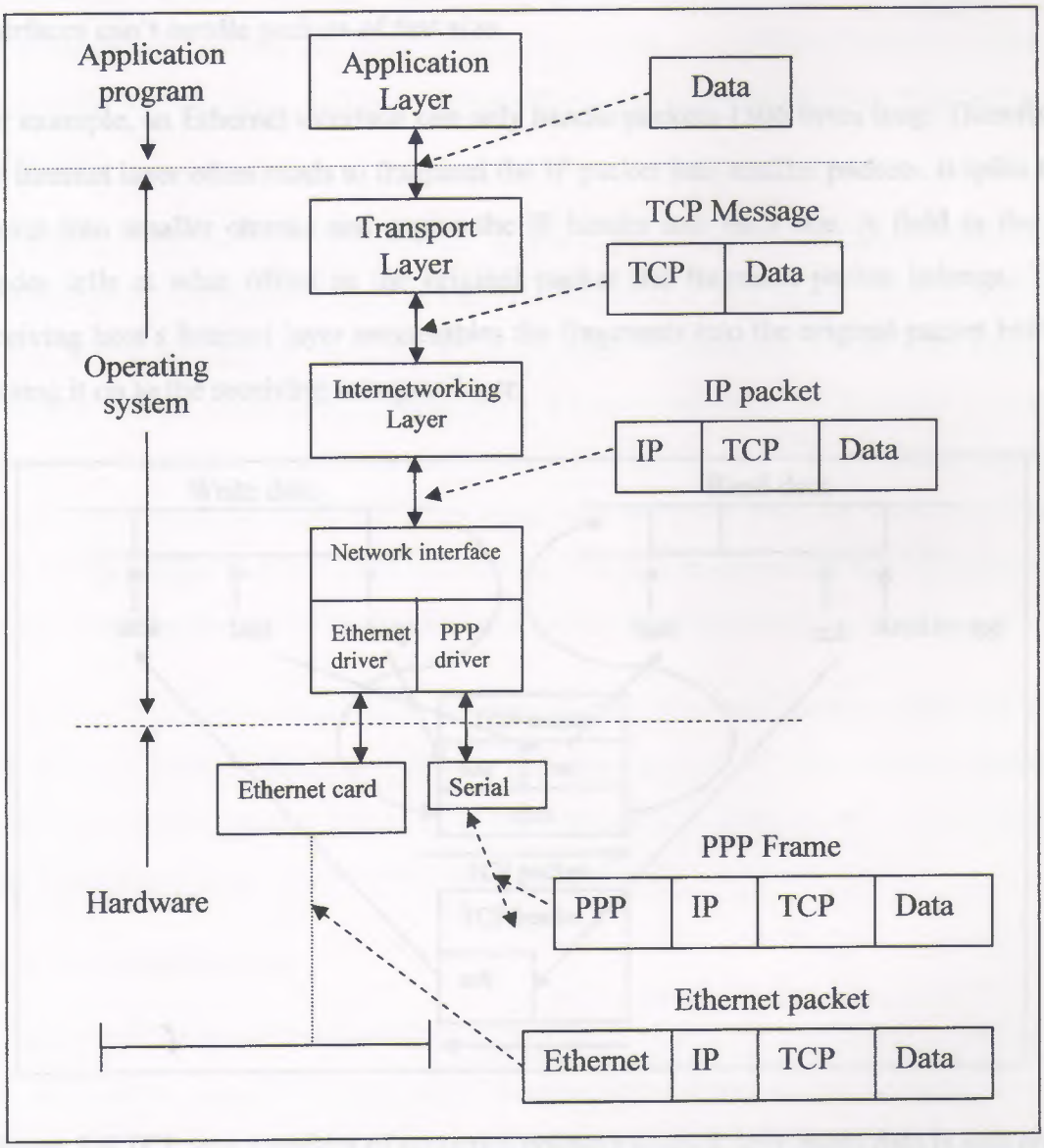


Figure 2.5 Each layer in a TCP/IP stack has a specific function and adds a header to the data from the application layer [13], [18].

Once the TCP header has been added, the TCP packet is passed to the Internet work layer. This layer gets packets to and from other hosts using the network interface. This

layer adds an IP header to the TCP packet. This header contains the destination host address and the source host address. These addresses along with the port numbers from the TCP header let us uniquely identify which TCP connection this packet belongs to as well as the direction it needs to goes.

The Internet layer needs to send the packet out over a network using one of the one or more network interfaces. Internet packets can be up to 8 KB long, but most network interfaces can't handle packets of that size.

For example, an Ethernet interface can only handle packets 1506 bytes long. Therefore, the Internet layer often needs to fragment the IP packet into smaller packets. It splits the packet into smaller chunks and copies the IP header into each one. A field in the IP header tells at what offset in the original packet the fragment packet belongs. The receiving host's Internet layer reassembles the fragments into the original packet before passing it on to the receiving transport layer.

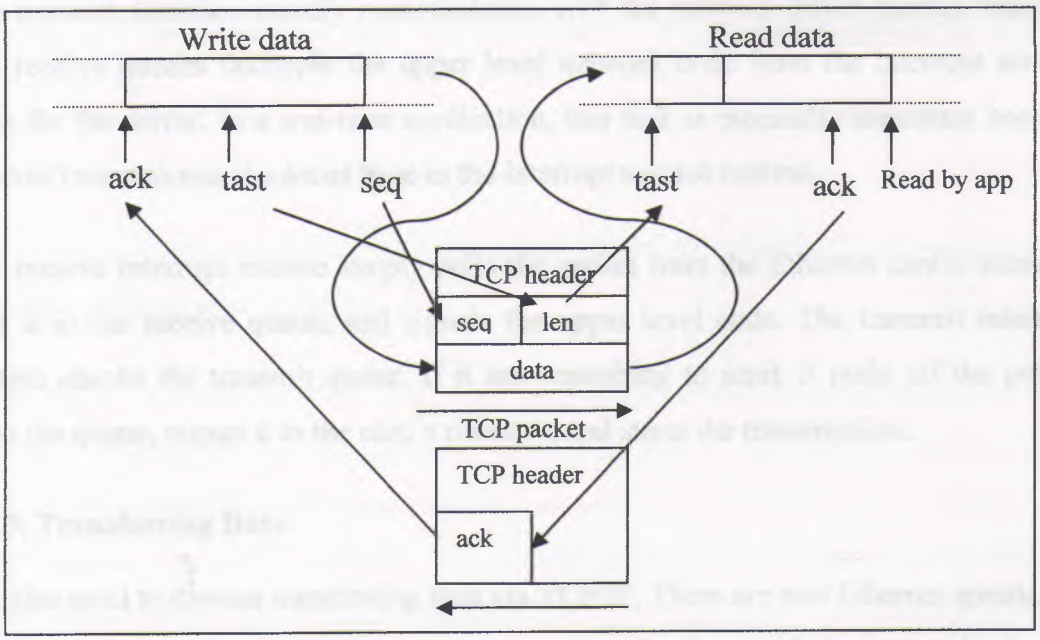


Figure 2.6 TCP uses a variety of sequence pointers to track how much data is sent and acknowledged out of the 32-KB window. TCP uses a sliding-window protocol, where each sequence number identifies a single byte.

For the Internet to work, we need to be able to connect various networks together via special hosts that have more than one network interface. These hosts are called routers

or Internet gateways. On a router, the Internet-layer implementation also needs to decide which network interface a packet needs to be sent on. A routing table does this job. Now we're at the business end of the protocol stack, the network interface layer, which has the device drivers for the network devices. A network device may be an Ethernet card, a serial port, or wireless Ethernet interface that connects the host to the network. Probably the most common network device is the Ethernet interface. Its versatile LAN architecture can be connected to twisted pair, coax, or fiber-based media. An Ethernet card usually has some memory in which the outgoing Ethernet packet is constructed and received. The actual transmitting and receiving is done with hardware on the card.

The network interface driver copies the packet it received from the Internet layer into the Ethernet card's memory, adds an Ethernet header to it, and tells the card to send the packet. When the card transmits the Ethernet packet, it interrupts the network interface driver. Of course, the card also interrupts if a packet was received from the Ethernet. The network interface usually communicates with the network driver queues, transmit and receive queues decouple the upper level network code from the interrupt service code for the driver. In a real-time application, this task is especially important because we don't want to spend a lot of time in the interrupt service routine.

The receive interrupt routine simply pulls the packet from the Ethernet card's memory, puts it in the receive queue, and signals the upper level code. The transmit interrupt routine checks the transmit queue. If it has something to send, it pulls off the packet from the queue, copies it to the card's memory, and starts the transmission.

2.4.3. Transferring Data

We also need to discuss transferring data via TCP/IP. There are two Ethernet speeds, 10 and 100 Mbps. Most100-Mbps cards automatically detect if the network they're plugged into is 100 or 10 Mbps. At 10 Mbps, Ethernet can transfer almost 1 MBps of data in one direction. However, you most likely won't see that high of a throughput. In many cases, if there's more than one host on the Ethernet, there will be some overhead in trying to access the media and deal with contention.

For Ethernet, this situation can be really bad. The utilization may be as low as 30% when Ethernet is saturated with tens of hosts.

That means all hosts together are only able to transfer about 300 kbps on a highly congested 10-MBps Ethernet. Of course, if only two hosts are involved, we should be able to achieve almost full utilization of the Ethernet.

Another problem is latency. To send data to another host, the data has to traverse the protocol stack, possibly get copied into the Ethernet card's memory, be transmitted over the Ethernet, be received and copied form the receiving card, and passed back up the protocol stack on the receiver and then reverse this to get an acknowledgment back.
On a lightly loaded 10-MBps network with high-performance Ethernet cards, you might see latencies down to 1.0ms. However, if the network gets loaded, the latency can easily reach 10–20 ms or more.

Basically, make sure the Ethernet is not very loaded if you expect high utilization or throughput. Also, this means there are upper bounds to what can be done with Ethernet as a device bus.

2.4.4. The command is of the form:

For every command received, the logic-engine interface sends back a byte again in a two-digit hexadecimal encoding.

I also haven't addressed how an Ethernet-based device finds out its Internet address, so it's easy to configure the Internet host address in one of the start-up scripts via the console. In an embedded Ethernet device, however, there probably isn't a console. But don't despair. There are solutions. One option: include a serial port on the Ethernet device, which would run a small command-line-based interface that lets you configure things like the Internet host address and store data in a flash file system. But, this solution mars the beauty of having an Ethernet device it would have a console and seem more like a computer.

A better solution is to use one of the protocols available for this purpose. There's the dynamic host configuration protocol (DHCP) and the boot protocol (BOOTP). The idea: you can use a Windows or Unix machine as an Internet host address clearinghouse.

When turned on, the Ethernet device sends a DHCP or BOOTP broadcast over the Ethernet. A workstation configured to be a DHCP or BOOTP server then assigns an Internet address to the Ethernet device and sends a response. The Ethernet device uses this address until it's turned off. It would be easy to replace the logic engine interface with a PC/104 A/D board that plugs into to the Ethernet device, then change the code to read out the data from the A/D board and send it to the workstation using a TCP/IP stream. Viola an Ethernet-based data-acquisition device, I'm can think of other applications as well.

2.5. Web Appliances and the Internet Protocols:

In order to understand the basic requirements of an Internet appliance, we must take a look at the manner in which the Internet works. Broadly speaking, there are those machines on the Internet, which serve up or deliver data on request, called servers and those machines requesting data from the servers, called clients. Each computer, whether server or client, connected to the internet gets a unique 32 bit identification tag called an IP address.

To avoid assigning a unique IP address to every terminal purchased or potentially able to connect to the Internet, Internet service providers generate dynamic addresses for every machine requesting an active connection. The same client connecting twice to the Internet may not be assigned the same address for each session. In contrast, servers, which host well known web sites, essentially require a fixed address. The allocation of these addresses is governed by agencies like InterNIC or IANA.

Raw data is passed around the Internet in the form of datagram's, specially marked packets, which get passed around from computer to computer. The data is split up into minor chunks of the above mentioned packets and encapsulated with headers and checksums to enable it to be routed to its final destination.

The Internet protocol used varies depending on the application. In most Internet applications, a dial up connection to an ISP is the first stage to gain access to the network layer (called the IP layer). This protocol is generally the point to point protocol, PPP, which will encapsulate all other protocols. Another protocol used to connect to the net is the SLIP, i.e. Serial Line Internet Protocol. PPP essentially encapsulates TCP and

IP packets and handles username and password authentication. The transport layer provides the basic connectivity between server and client. In this layer, we can choose between a simpler protocol called UDP (User Datagram Protocol) and a more robust TCP (Transmission Control Protocol). UDP is not based on making a connection between two hosts, it is a connectionless protocol, i.e., the receiving host does not acknowledge the packets it receives from the host and thus the transmitting host has no indication as to whether the datagram has made it to the receiving computer.

UDP provides best effort delivery of data with an optional checksum to preserve data integrity. UDP packets have the same reliability as IP; packets are not guaranteed to be received in order at the remote host. UDP packets can be sent at full speed-as fast as the underlying physical device can send them. On slow processors, UDP's lack of overhead can make a large difference in the throughput when compared to TCP. On fast processors, the difference is not as large. The lack of an end-to-end connection in UDP means that it can be used to send one-to-many and many-to-many type messages.

In contrast, TCP offers connections based and flow controlled transfer of data. TCP is generally accepted to be the more reliable of the two protocols, but the choice in the case of our web appliances is influenced by considerations of the type of data being dealt with. For example, consider a device transmitting ambient temperature values measured on site over the net. The device establishes a connection with the server every few minutes and uploads data. Loosing a few readings may not be as severe a handicap in this case; temperature, which changes, but slowly, can easily be interpolated and found. The simpler UDP in this case comes close to achieving some of the features of TCP.

Before sending data to a remote host, a TCP connection must be established. This means that only data between two end-hosts may be exchanged over a TCP connection. Every TCP segment that contains data is acknowledged to provide reliability. The acknowledge segments themselves are not acknowledged to prevent an infinite recursion. TCP's reliability comes at the price of being much more complicated than UDP [18].

In order to negotiate Internet protocols, the Internet appliance uses a Network Stack. A stack can be a software implementation, commonly available with many vendors. However, TCP has large RAM and ROM requirements to keep track of connections and packets not acknowledged by the remote computer (TCP has a habit of retransmitting lost datagram's). A software stack implementation may run into stonewalls in such cases, especially since the hardware implementation of a web appliance generally is centered around a microprocessor, able to address limited amounts of memory and in addition having to control the working of the appliance in itself. Packing this sort of functionality into a design may require tradeoffs, cutting down on some options, which could otherwise be available to the user etc. Embedded designers have many different concerns-for some, code size is of ultimate importance, while for others, data rate and reliability are a higher priority. The simplicity of UDP is slightly misleading-UDP applications typically require code at the application layer to build in some level of reliability. Data sequencing is the most common addition because it allows the application to mark missing data and to resolve out-of-order packets.

The case study we present uses a hardware implementation of a network stack, which relieves large amounts of memory to be left for other code. The block diagram shows the hardware components required for constructing a low cost arrangement like this, which can double as a web server as well as a client. The processor choice is wide, with a playing field split between several manufacturers. The device of interest is the hardwired Network stack the S7600, manufactured by Seiko. The hardware stack must minimally provide a Network access layer implementation, Internet layer and transport layer as shown. Once configured the chip acts as a data buffer, storing data on its 1024 byte internal RAM while its TCP engine appends various checksums and headers to it. The chip has registers to store the Internet addresses of both the server as well as client and supports PPP as well as Password and Username authentication. The only code the designer has to write to web enable the device is limited to initializing the registers to necessary values. No software routines as to the computing of checksums or framing of datagram are needed. Consequently the code size works out to be much lower than in case of a software implementation. A suitable modem is to be chosen. For our temperature application mentioned above, it is quite obvious that speed is not of prime consequence and hence a lower cost modem capable of only lower speeds can be used

as well. Choosing a microcontroller that provides on chip flash memory further can shrink the size of the device construction.

2.6. Web Appliances and their Implementation:

The instrument collects data on the temperature using sensors and front ends, which are not shown. The objective is to get the data to a remote terminal for analysis. The instrument can either operate in server mode, where a remote computer can connect to it, to access data, which is served up by the instrument in the manner of a web server. The other alternative is to configure the instrument as a web client, wherein it will connect to a site on a web server and upload data. The remote machine can then access this data by requesting the site with a standard web browser interface.

A third alternative also exists, wherein the instrument is configured to send data periodically via emails to any desired email ID. Such a device configuration is called, among other things an e-mail-reporting engine. We will focus on the client operation of the device. The flow chart for client operation is shown. The client application needs to know the port number and IP address of the server it is meant to upload data to. This means that there must be a program running on the server end that will listen on the above-mentioned port. Once connected the transfer of information may take place between server and client.

The user requesting data connects to the server using a simple web browser. The responses sent back to the browser are in the form of HTTP responses. Broadly, there are five methods of delivering data through HTTP [20].

- Static html pages.
- Run time generated HTML.
- Scripting languages.
- Stylesheets.
- Object interfaces.

Static HTML pages can be used to display information that does not change. They can be stored as files in file systems or simply in memory without a file system. Dynamic HTML pages are generated during application run time.

For example C functions can be used to create these pages in memory. The function can be thus that the pages displayed show the latest data. Scripting gateway interfaces are similar to run time generated HTML except that an interpreted scripting language is used to HTML pages. The choice of scripting languages is split between PERL, JavaScript and ASP. Stylesheets have been used to transform data files into HTML pages. Object interfaces are applications that can run in a browser like Java applets and active X controls. Object interfaces are available from the server upon browser request. The speed at which the server can deliver data to the browser depends upon the mode chosen and ranges from Static HTML (fastest) to Objects (slowest). Apart from these there is another method that is of essence if the server happens to be a JAVA server. Servlets are yet another offering from the growing pool of JAVA based solutions. A Servlet is a JAVA based program analogous to an APPLET, but where an APPLET needs a Virtual Machine or a JAVA supporting browser, a SERVLET runs on the server end of our application. Our Servlet opens a specific user defined port (the lower 1024 ports that can be opened for client server communications are reserved for certain well known services) for the web appliance and listens for a request from the web browser. Such an HTTP request will initialize the Servlet. The Servlet then reads in the data from the web appliance and send it to the requesting browser in the form of well-formatted HTML/TXT responses. The reloading of the page periodically is necessary to refresh data, the page being essentially static in nature. Quite obviously this works only because the data that is being uploaded is of slowly changing nature. Our Web appliance can use this technology. The embedded device which is a client creates a client socket and the Servlet creates a Server Socket at the server end. As the user requests for the data from the Web appliance to be seen, the Servlet is invoked through the browser and a snapshot of the current data can be uploaded to it. The response is in HTML form and is forwarded to the client (i.e. the browser) on which it is displayed.

The hardware consists of a microcontroller, a network stack and a modem. The microcontroller performs the functions of collecting data on the quantity to be measured, say temperature, and also is responsible for configuring the network stack and modem. The S7600 needs to be initialized before it goes online. The microcontroller sets the clock divider rate and writes modem configuration to the serial port buffer of the S7600. The control of the serial port is given over to S7600.

By writing to PPP registers, the S7600 can be forced to bring up a PPP connection to an ISP. PAP is optional and depends on whether the ISP uses password authentication. The S7600 can be set to function in TCP server or client mode and also supports a UDP mode. The stack has registers to hold the 32 bit IP Addresses for the remote server as well as for itself. If the ISP assigns addresses dynamically, the address is negotiated for in the PPP phase. Port registers determine which port to open for communication, as mentioned above, we must exercise some care in the choice of port numbers, since some are assigned to fixed services.

The socket registers allow us to open sockets and read to or from them. It is quite obvious that the entire exercise of the controller interfacing to the Internet reduces to merely pushing 8 bit data to our network stack. The modem can be an ordinary external modem, the sort that goes to the serial port of the ubiquitous PC or it can also be an embedded modem which can be fabricated right on the instrument board itself. The modem need not support very high baud rates. The Si2400 chip set is a good example of an embedded modem. The embedded modem usually supports a working subset of the complete Hayes commands that a complete external modem supports. AT commands are sent as ASCII strings to the modem.

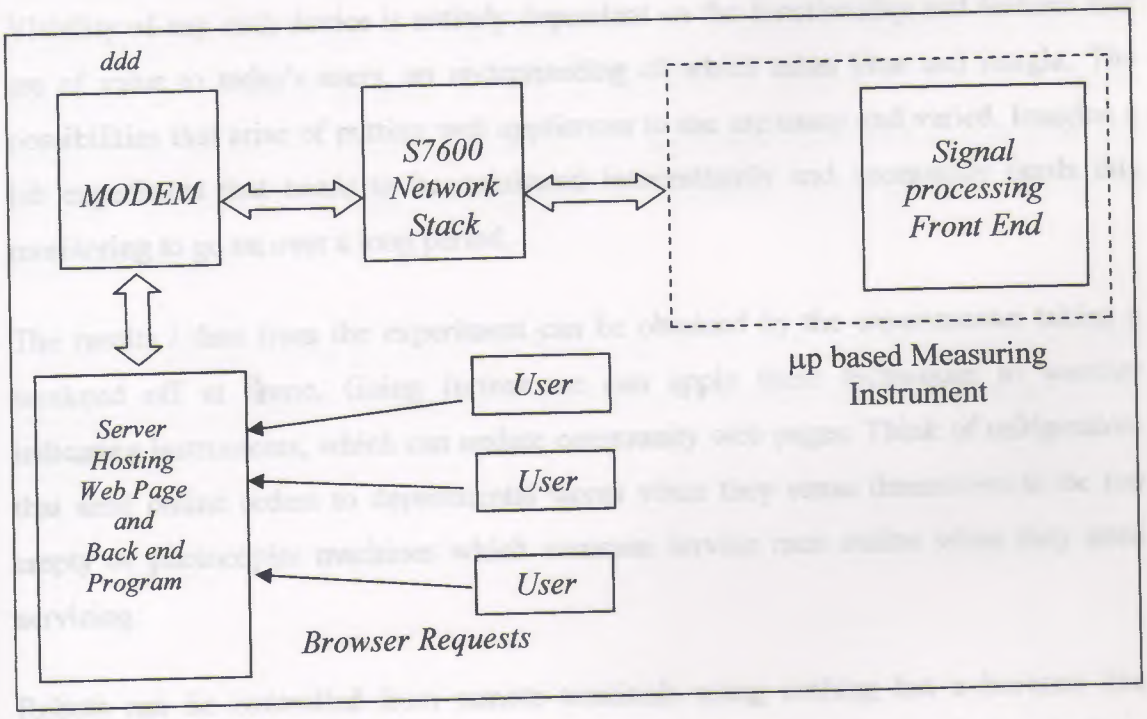


Figure 2.8 Concept of Web Appliance

The Systems software in this application first initializes the Network Stack and the Modem. Modems have different initialization strings, best referred to from their individual manuals. The first step in establishing the dialup is forcing the modem Off-Hook, then with a small delay to wait for the dial tone and then we set the PPP register in the Network Stack to kick start the PPP dialup. Immediately after this we send dial string (ATDT<phone no of ISP>) to the Serial port buffer of the Network Stack. The processor relinquishes control of the serial port to the S7600. Once the PPP has been established PAP request is sent to the ISP. The Username and Password follows the PAP request [19].

The Network Stack is ready to open Sockets on the client and server and hence data transmission commences. Some form of Server-side programming in the form described above is necessary. Beware, however not all servers allow you to open sockets indiscriminately on them!

2.7. Scope of Web Appliances.

Having taken a look at the engineering of a web appliance i now consider its feasibility or viability in today's networking context.

Viability of any such device is entirely dependant on the functionality and features that are of value to today's users, an understanding of which takes time and insight. The possibilities that arise of putting web appliances to use are many and varied. Imagine a lab experiment that needs to be monitored intermittently and necessarily needs this monitoring to go on over a long period.

The results / data from the experiment can be obtained by the experimenter taking a weekend off at home. Going further we can apply these techniques to weather indicating instruments, which can update community web pages. Think of refrigerators that send online orders to departmental stores when they sense themselves to be too empty or photocopier machines which summon service men online when they need servicing.

Robots can be controlled from remote terminals using nothing but a browser like interface. Web enabled laser printers, which can receive and print files directly from

email attachments could be serious competition for an overpriced fax system. One could combine the quality of superior print with the rock bottom prices of sending emails.

However it is also equally true that the basic nature of appliances requires some compromise of cost vs. feature vs. size vs. portability, but decisions concerning what features the consumer expects and desires will separate commercially successful appliances from those which can be considered of little consequence in today's market.

2.8. Summary

This chapter presented the commonly used internet protocols and also describes how the web appliances can be designed using these protocols.

IP addresses provide the foundation for identifying individual network interfaces (and therefore computers or other devices as well) on TCP/IP networks.

Understanding address structures, restrictions, and behavior is essential to designing TCP/IP networks and appreciating how existing TCP/IP networks are organized.

Understanding binary arithmetic is essential to knowing how to deal with IP addresses, particularly when working with subnet masks, knowing how to convert from decimal to binary, and vice versa, help you understand how the concept of stealing bits from the host portion of an IP address permits a network to be subdivided into logical subnetwork, or subnets. Likewise, it helps you understand how stealing bits from the network portion of multiple contiguous IP addresses increases the number of addressable hosts.

I have actually give you guidelines to build one & to show you how many different options does a designer charged with the task of including web connectivity in his design have?

3. MICROCONTROLLER CRYPTOGRAPHY

3.1. Overview

When systems communicate telemetry or control information between peers the security and authenticity of the communicated data may be important. If the medium is public or could be subject to compromise, securing the communications path becomes an issue. But encrypting control and status messages that are passed between subsystems on networks, telephone lines, or RF channels usually requires extensive microcontroller resources, and the maintenance of secrets (keys) is usually the weak point in the system. Changing or customizing critical system secrets is often impossible in ROM-based equipment, which further reduces the security of the system. *Dallas Semiconductor* manufactures low cost *1-*Wire® memory devices that contain fast, powerful cryptographic engines. Some of these devices have the ability to perform the SHA-1 hash very quickly, and to securely store, protect, and rotate secrets. These devices can be used with small microcontrollers and limited resources to provide strong small-message encryption and peer-to-peer authentication between subsystems.

Security features are useful if an application dispenses services on a pay per service basis. Electronically bypassing the security would allow the dispensing of the service for free, resulting in lost revenue to the system owner. Another common application is the transmission of secret information. The user's algorithm and key data could be observed in an unsecured system, resulting in a break in the secure transmission. The Secure Microcontroller Family is designed to protect the contents of memory from being viewed. This is done with a combination of circuit techniques and physical security. The combination is a formidable defense. Regardless of the application, the secure microcontroller protects the contents of memory from tampering and observation. This preserves secret information, access to services, critical algorithms etc. The security features of the Secure Microcontroller include physical security against probe, memory security through cryptographic scrambling, and memory bus security preventing analysis of the CPU's operation. The features mentioned above and described below protect the application code and data.

3.2. Enhancing Embedded Security

More and more embedded systems are connecting to company intranets or to the Internet. Such network connections can greatly enhance an embedded system's utility and capability. Unfortunately, they also increase the system's vulnerability to attack by leaving an open door for malicious programs to enter. To enhance the security of these networked embedded systems, operating systems must change.

Embedded systems with network conditions find use in an ever-increasing range of applications. Industrial control systems use networks of embedded control nodes for applications such as chemical processing, electrical-power distribution, and factory automation. Entertainment systems, such as set-top boxes and game consoles, use the Internet for downloading new menus, features, and games. Home control systems are using both the Internet and their own intranet to manage heating, lighting, and security for residences. Even household appliances are connecting to networks to add features and automate maintenance. A common aspect of all these applications is that the embedded system uses the network as a channel for receiving instructions, control parameters, and new programs [50].

For embedded systems that connect to the Internet, this openness means that virtually anyone can gain access to the system. If the system allows software updates, abusers can load malicious programs into the system and cause it to misbehave or crash. Even if changing a program is impossible, an abuser can enter invalid or inappropriate parameters and commands.

3.2.1 Encryption Is Not Enough

Implementing encryption protocols would seem to address the problem, but encryption has several drawbacks for embedded systems. For one, encryption is a computationally intensive activity that may be impossible in smaller embedded systems. For another, the protocols may be ineffective in an emb

edded system.

Key encryption method would be ineffective. Every developer would need to know every game system's private key, so the key would hardly be private. A public-key system, on the other hand, leaves the system open to attack from someone mimicking a game developer.

Most embedded systems use RTOSs (real-time operating systems), which allow application programs to fully access system resources.

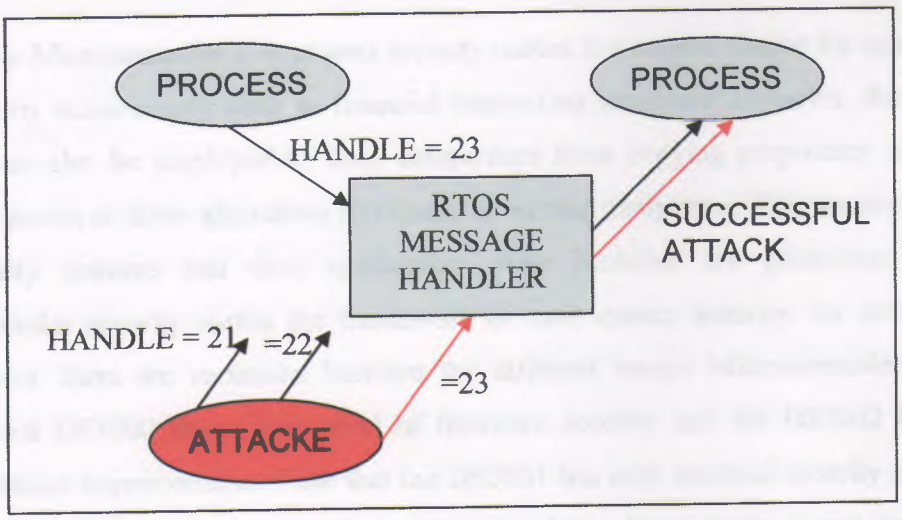


Figure 3.1 By duplicating the "handle" used to identify a message's recipient in Interprocess communications, a malicious process can disrupt the behavior of an embedded system.

3.2.2. Attacking Interprocess Communications

A more subtle attack mechanism is to interfere with a system's interprocess communications. Most RTOSs use messaging queues to transfer information from one process to another. The information transfer requires the sending process to make a system call to the RTOS (Figure 3.1). The call provides the RTOS with a pointer to the memory location where the information resides, along with a handle, or identifying number that specifies which process is to receive the message. The operating system then notifies the receiving process of a pending message [50].

3.3. Firmware Security

One of the most unique features of the Secure Microcontroller is its firmware security. The family far surpasses the standard offering of ROM based microcontrollers in keeping system attackers or competitors from viewing the contents of memory. In a standard EPROM based microcontroller, a knowledgeable attacker can disable the EPROM security bit and have access to the entire memory contents [35].

The Secure Microcontroller's improved security makes it a natural choice for systems with high security requirements such as financial transaction terminals. However, the firmware security can also be employed to keep competitors from copying proprietary algorithms. Allowing access to these algorithms can create an instant competitor. This section describes the security features and their application. Also included are guidelines to using microcontroller security within the framework of total system security. As with memory map control, there are variations between the different Secure Microcontroller versions. The original DS5000 has a high level of firmware security and the DS5002 has added several distinct improvements. Note that the DS5001 has only minimal security and should only be applied when other physical security is used or when security is not needed. The table 3.1 provides a brief summary of the versions and their security features. A detailed description of each feature follows. In the description, elements that are unique to a particular Secure Microcontroller version have that version underlined.

Table 3.1 Summary of Security Features of Particular Secure Microcontroller

FEATURE	DS5001	DS5000	DS5002
Security Lock	Yes	Yes	Yes
RAM memory	Yes	Yes	Yes
Encrypted memory	None	Yes, user must enable	Yes
Encryption	Key None	48 bits	64 bits
Encryption Key Selection	None	User selected True	random number
Encryption Keys loaded	N/A	When user selects	Automatic, any new

			load, dump
Dummy bus access	None	Yes, when encrypted	Yes
On–chip Vector RAM	None	Yes, when encrypted	Yes
Self–Destruct Input	None	None	Yes
Die Top Coating	None	None	Optional (DS5002FPM)
Random Number Generator	Yes	None	Yes

3.3.1. Security Lock

Ordinarily, the easiest way to dump (view) the memory contents of a Secure Microcontroller is using the Bootstrap Loader. On request, the Loader will transfer the contents of memory to a host PC. This is prevented by the Security Lock. The lock is the minimal security feature, available even in the DS5001. Once set, the Security Lock prevents the Loader from gaining access to memory. In fact, no Loader commands except Unlock) will work while the Lock is set. The Security Lock is similar in function to an EPROM security bit on a single chip microcontroller. It prevents a programmer from reading the memory. In addition, the Security Lock prevents the microcontroller from executing code on the expanded bus of Ports 0 and 2. Thus an attacker can not add a memory and use MOVC instructions that would force the microcontroller to read out the contents of protected memory. However, the Secure Microcontroller Security Lock does provide one important difference from EPROM security bits. When the Security Lock is cleared, it destroys the RAM contents. If a knowledgeable user were to physically erase the security bit in an EPROM–based microcontroller, the memory contents would remain to be read. The Security Lock consists of a multiple bit latch distributed throughout the microprocessor with circuits that collapse the lock in the event of tampering. Clearing the lock starts an irreversible destructive process that acts differently for each device as described below. In a DS5001 clearing the lock causes the loader to manually write over the first 32K bytes of NV RAM with zeros. Thus the contents of memory would be erased. This is obviously a low level of security but would deter casual inspection. In a DS5000 or DS5002, clearing the lock causes an instantaneous erasure of the Encryption Key and

Vector RAM. This action is unpreventable once the lock is cleared and happens independent of VCC or crystal. Once the erasure has occurred, a DS5000 assumes a non–secure (brand–new) state. In a DS5002, the Loader proceeds to load a new Encryption Key once the erasure has occurred. In both, the Bootstrap Loader will then proceed to overwrite the first 32K bytes of RAM if power is available and the crystal is still present. This last action is for thoroughness. In systems that really require security, the Lock should be combined with Memory Encryption (discussed below).

3.4. How to Design a 3DES Security Microcontroller

Using IP cores and a pre-integrated IP platform, engineers at SoC Solutions built a custom microcontroller platform in less than two modules. Today's seemingly limitless access to information has also brought forth a need to secure personal and corporate information from unauthorized access and to protect privacy. This need for secure data not only applies to securing wired and wireless communications, but is also important in applications where access control; data integrity, confidentiality, and authentication are required. For this reason, cryptography will find its way into a host of common devices, including bank ATMs, kiosks, information portals, video surveillance equipment, building access controls, and the like [51].

3.4.1. 3DES Secure Data Encryption/Decryption Application

Many applications, such as online private information transfers, require that data files or streamed information be secured by encrypting the payload data. 3DES is considered to be very secure because it can use three separate keys to encrypt data. Our secure microcontroller implements a smart device that stores DES encrypted data in memory; it then decrypts the data before storing it back to memory. This is useful where data is provided as medium to large data files (or streams) and where the microprocessor has limited bandwidth to do the math intensive encryptions (or decryption) while simultaneously processing other tasks or applications [51].

The 3DES core uses a three-key concatenated DES algorithm to provide 192 bits of security. Keys can be stored in memory or input through a software application, which is

run on the secure microcontroller's microprocessor core. The microprocessor controls data movement to and from memory.

It also sets up the 3DES engine. Extra microprocessor bandwidth is available for additional application software. We used a 32-bit microprocessor, although a Xilinx Micro Blaze™ core could also be used in our secure microprocessor design. Figure 3.2 is a flow diagram of an encryption/decryption microcontroller device.

3.4.2. Reference Design

The two main hardware components of the reference design are the microprocessor core chip and a platform FPGA, such as a Xilinx Spartan™ or Virtex device. All of the functions (excluding the microprocessor) are implemented in the FPGA using SoC Solutions' IP platforms and soft cores. The secure microcontroller reference design provides the basis, or starting point, for the custom design. This implementation uses the microprocessor to load memory, control the start of encryption/ decryption, and then verify the stored result. In a typical application, a communications port such as Ethernet, PCI, USB, or a simple COM port would be used as a data I/O port. The design includes a direct memory access engine to read and write data from/to the 3DES core to internal FPGA SRAM or external memory. Also included are an internal SRAM controller, an interrupt controller, timers, UARTS, and an external bus interface. The process of encrypting or decrypting a file is simple. The microprocessor fills memory from data acquired either under software control or through a COM port. The microprocessor then sets up the DMA engine with a source address, destination address, and a block length. The processor writes a start bit to the DMA engine and then "lets 'er rip." The DMA engine reads a sub-block from the source address, sends it to the 3DES core for processing, and then writes the processed sub-block results to the destination address.

Timers can be used to poll the DMA engine dma_done flag to know when the operation is complete. The interrupt controller can also be used to signal the end of the 3DES operation.

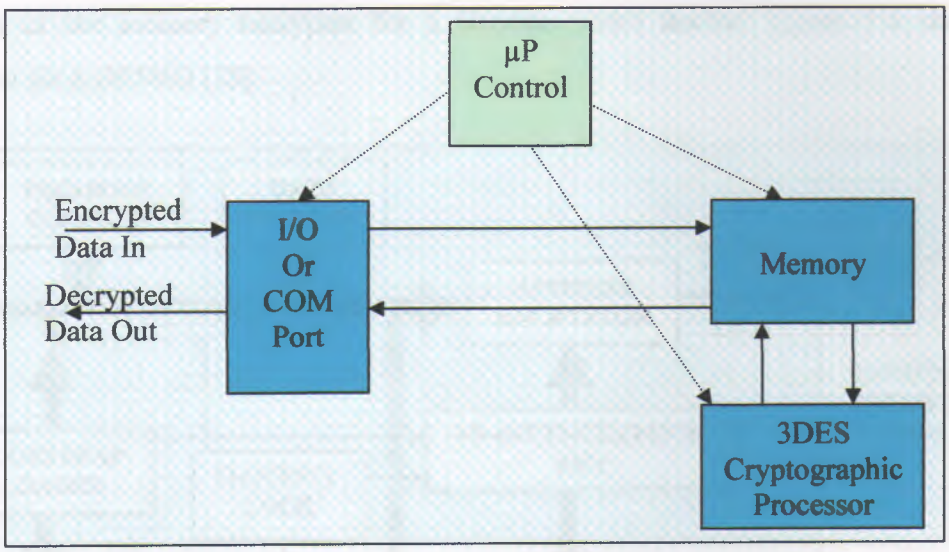


Figure 3.2 Secure microcontroller data flow diagram [51].

3.5. RAM Memory

NV RAM provides a useful way to store program and data. The contents can be retained for a long period, but can be changed when desired. This attribute is important when considering security. No matter what probing techniques are used on a ROM, the contents remain unaffected. With resources and patience, a determined attacker will obtain the contents of a ROM based product. NV RAM can be destroyed on demand. The user's physical security must simply remove the power (VCC and VBAT) from a microprocessor chip to eliminate the memory contents. Thus NV RAM provides flexibility as well as security. Enough physical security can be combined with even a DS5001 to provide a very secure system. The DS5002 even provides a direct facility to destroy memory discussed below [35].

3.5.1. Encrypted Memory

The heart of Secure Microcontroller security is the memory encryption function. Since the NV RAM is visible, the memory contents and memory bus are encrypted. That is, in real time, the addresses and data moving between the RAM and the microcontroller are scrambled by on–chip encryption circuits. Thus an attacker that observes the RAM contents or memory bus will see unintelligible addresses and data. Figure 3.3 shows the conceptual

diagram of the memory encryptor for a DS5000 series device. Figure 3.4 shows the encryptor for a DS5002 [35].

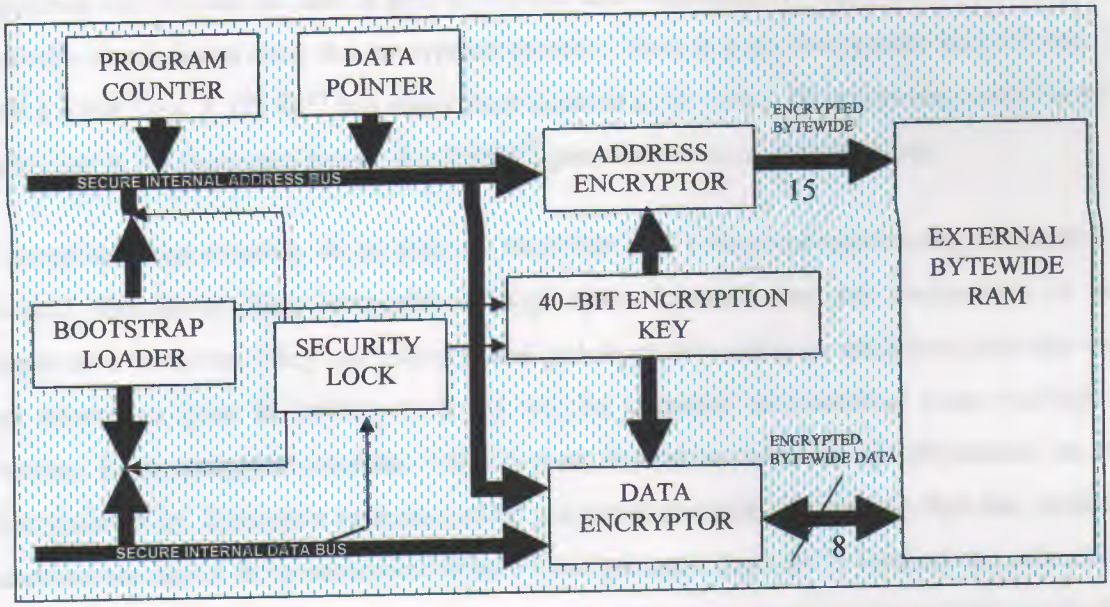


Figure 3.3 Ds5000 Software Encryption Block Diagram

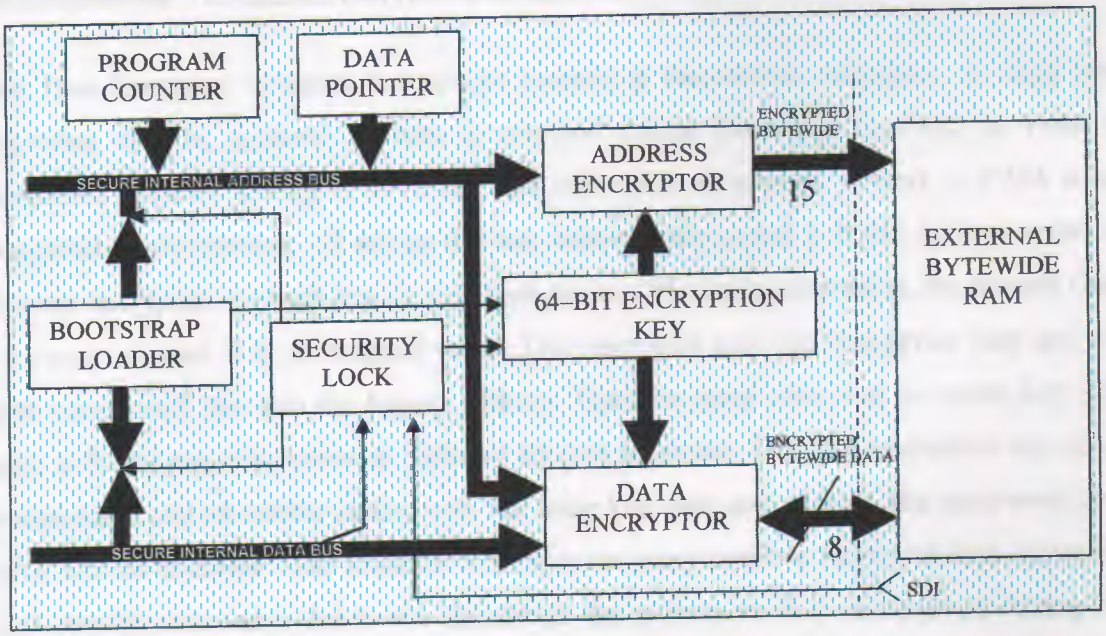


Figure 3.4 Ds5002 Software Encryption Block Diagram

In a DS5000, the encryption feature is optional. A DS5000 can be locked irrespective of its encryption and encrypted irrespective of the lock. Neither makes much sense by itself. The

encryption process is enabled by loading an Encryption Key for the first time. Prior to loading a Key, the DS5000 remains in a non–encrypted state. Once encrypted, the memory interface will remain so until a part is locked, then unlocked. The process of clearing the Security Lock deactivates the encryption circuits. Note that an Encryption Key of zero is still a valid Key. A DS5002 has encryption enabled at all times. No extra steps are required to invoke it. As discussed below, the DS5002 generates its own security Keys.

Encryption logic consists of an address encryptor and a data encryptor using separate but related algorithms. This encryptor is high speed circuits that are transparent to the application software. They are bidirectional and repeatable. That is, addresses and data that are scrambled prior to writing to RAM will be correctly unscrambled when reading in reverse. Each encryptor operates with its own algorithm but both are dependent on the Encryption Key. Encryptor operates while programs are being loaded so that the memory contents are stored in its scrambled form. When program memory is fetched, the process is reversed. Thus the actual program or data is only present in its "true" form while inside the microcontroller. The address encryptor translates each "logical" address.

The Data Encryptor operates in a similar manner to the address encryptor. As each byte including opcode, operand, or data is received during Bootstrap Loading, its value is scrambled prior to storing it in RAM. The value that is actually written in RAM is an encrypted representation. All values that are subsequently stored in RAM during execution also are encrypted. As each byte is read back to the CPU during execution, the internal Data Encryptor restores it to its original value. This encryptor uses the Encryption Key and the data value itself, but also the logical address. Thus the same data with the same Key will have different physical values at different address locations. The data encryption algorithm is repeatable and reversible so that with the same key, data and address, the same encrypted value will be obtained. Note however that there are many possible encrypted data values for each possible true value due to the algorithms dependency on Key and address. Using the combination of address and data encryption, the normal flow of program code is unintelligible in the NV RAM. What had been a sequential flow of addresses is now apparently random. The values stored in each memory location appear to have no relation

to the original data. Another factor that makes analysis more difficult is that all 256 possible values in each memory are valid possibilities. Thus an encrypted value is not only scrambled, but it becomes another potentially valid byte.

Different memory areas are encrypted in the DS5000 and DS5002. For a DS5000, all memory accessed under CE1 can be encrypted. CE2 is not encrypted. This allows access to peripherals such as a Real–time Clock to be performed using CE2. For the DS5002, encryption is performed on all bytes stored under CE1 through CE4. The memory or peripherals accessed by PE1 through PE4 on a DS5002 are not encrypted.

3.5.2. Encryption Algorithm

The Secure Microcontroller family uses a proprietary algorithm to encrypt memory. The DS5000FP and DS5002FP use different encryption algorithms. They are the result of improvements made over time in the proprietary encryptor circuits. The original S5000FP (circa 1988) has the first version of encryptor. This was soon improved with a second version encryptor in 1989, and remains in production today. A substantial improvement was made in the DS5002FP, which uses a wider Key and a more non–linear algorithm. The DS5002FP memory encryptor uses elements of the DES (Data Encryption Standard) although not the entire algorithm. Full DES is impractical as memory encryption must be performed in real–time on a one–to–one substitution and not a block cypher basis. The encryption algorithm is supported by the fact that both address and data are encrypted, the algorithm and key are both secret, the most critical data can be stored on chip in vector RAM (discussed below), and the bus activity is scrambled using dummy access (discussed below). For this reason, a security analysis of the DS5002FP is not simply a mathematical treatment of the encryption algorithm.

3.5.3. Dummy Bus Access

The Secure Microcontroller makes its memory contents obscure through encryption. Additional steps are also to prevent analysis of the bus activity by 8051–familiar hackers. Both the DS5000FP and DS5002FP insert dummy memory operations when possible. In the 8051 architecture, there are typically two identical memory accesses per instruction

cycle, but most operations so nothing with the second program fetch. In the Secure Microcontroller, a pseudo–random address is generated for the dummy cycle and this random memory address is actually fetched, but the dummy data is discarded. The orders of the real and dummy accesses are switched according to a pseudo–random process. This is repeatable so that the execution always appears the same. During these pseudo–random cycles, the RAM is to all appearance read. Thus by repeatedly switching between real and dummy access, it is impossible to distinguish a dummy cycle from a real one. In analyzing bus activity, a large percentage of the memory fetches will be garbage that has no meaning. The dummy accesses are always performed on a DS5002FP, but are only used on a DS5000FP when encryption is enabled. Naturally, dummy accesses are always read operations since the dummy address might contain valid data.

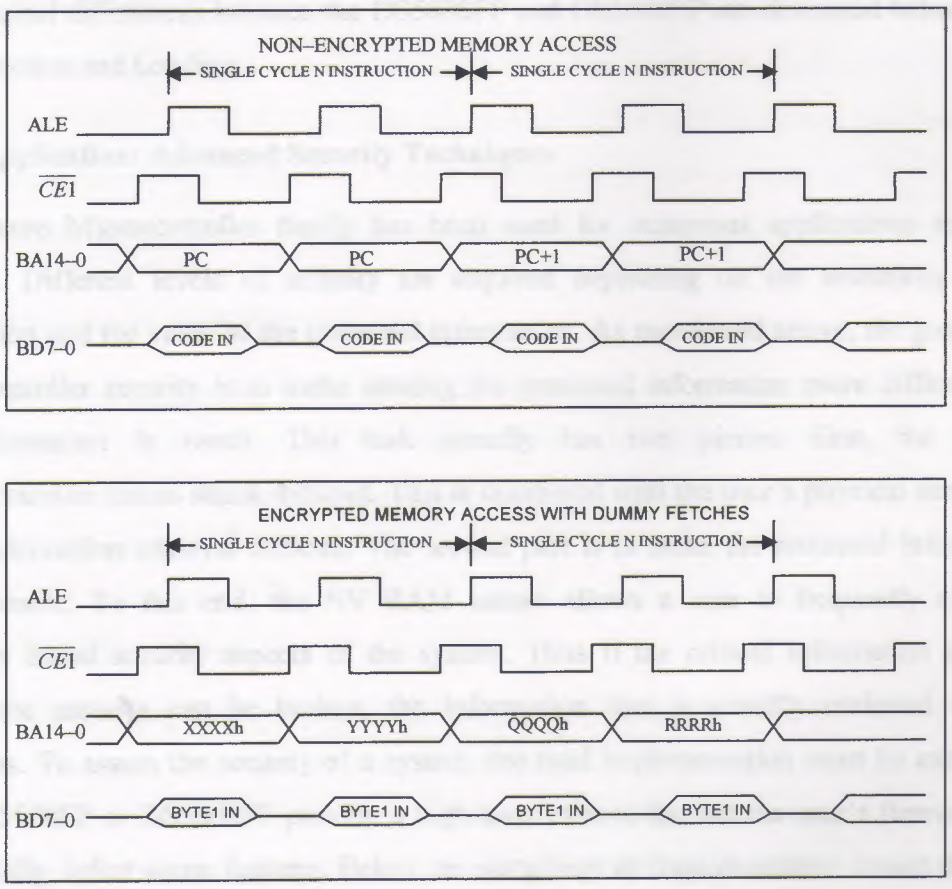


Figure 3. 5 Dummy Bus Access Timing

3.5.4. Encryption Key

The DS5000FP uses a 40–bit Encryption Key that is stored on–chip. As mentioned above, the Key is the basis of the encryption algorithm. The resulting physical addresses and data are dependent on this value. Tampering with or unlocking the microcontroller will cause the Key to be instantaneously destroyed. If the memory contents are encrypted, they become useless without this Key. A user selects the 40–bit Key and loads it via the Bootstrap Loader. Selecting this Key enables the encryption feature. The DS5002FP uses a 64–bit Key. It is similarly stored on–chip in tamper resistant circuits. In much the same way, this Key is the basis for the physical values that are presented on the bus. Using a wider Key gives the encryption more complexity and more permutations that must be analyzed by an attacker. Apart from the width of the Key and complexity of the encryptor, the principal differences between the DS5000FP and DS5002FP are discussed below under Key Selection and Loading.

3.5.5. Application: Advanced Security Techniques

The Secure Microcontroller family has been used for numerous applications requiring security. Different levels of security are required depending on the sensitivity of the application and the value of the protected information. As mentioned above, the goal of the microcontroller security is to make stealing the protected information more difficult than the information is worth. This task actually has two pieces. First, the Secure Microcontroller makes attack difficult. This is combined with the user's physical security to make information retrieval difficult. The second part is to make the protected information less valuable. To this end, the NV RAM nature allows a user to frequently alter the firmware based security aspects of the system. Thus if the critical information changes before the security can be broken, the information that is actually retrieved will be worthless. To assess the security of a system, the total implementation must be examined. The DS5000FP or DS5002FP provide a high level of security, but the user's firmware can accidentally defeat some features. Below are samplings of implementation issues that will make the DS5000FP or DS5002FP more difficult to crack. There are also suggestions on making a system more secure using external circuits.

3.5.6. Avoid Clear Text

The encryption algorithms used by DS5000FP or DS5002FP are generally adequate to prevent analysis when combined with well developed code. However, the encryption is defeated to some extent if the user stores text that appears on a display in encrypted form. This gives the pirate a starting point to look for the clear text in encrypted storage and analyze the encryption algorithm. The "data answer" is already known. If clear text is required, then preferably store it in no encrypted memory. If this is impractical, then disperse it so that it is hard to find. Avoid at all costs reading the clear text from memory then immediately displaying it. This is a sure means to identify the encrypted values of the text for the attacker.

3.6. Avoid CRC or Checksum

Running a checksum on power up provides the pirate with a sequential listing of the addresses in encrypted form. Therefore the attacker has a great advantage in deciphering the Address Encryptor. Preferably avoid a checksum. If one is needed, then check the minimum amount of memory and perform the check in non–sequential fashion.

3.7. Avoid Long Straight Runs of Code

A common coding practice is to run numerous sequential operations. This is common knowledge and should be avoided. The pirate can use this in the same way as a checksum process. It provides a sequential listing of encrypted addresses and assists with analysis of the address encryption.

3.8. Use Random values

The Random Number Generator of the DS5002FP can be used to make a pirate's task more difficult. When time is available, the software should perform random actions at random time intervals. As an example, the Random Number Generator can be used to select a timer interrupt value. Thus the microprocessor will be interrupted at random intervals making characterization very difficult. Software can elect to out of Vector RAM for a random period of time. Also as discussed above, the microprocessor generates dummy RAM reads

when possible. However, it can not generate dummy writes. However the user's code can. Random numbers can be written to address that are known to be unused. If this is done while the microprocessor is visibly performing a meaningful task, it will make analysis very difficult.

3.9. Change Code

Perhaps most importantly, the user should reprogram portions of the Secure Microcontroller that deal with security. For example, if the microprocessor is performing DES, the user can change DES keys. Any security system can be broken with enough time and resources. By altering the security features, this threat can be minimized.

3.10. External Circuits

A variety of external circuits can support secure operation. For example, the DS2400 is a unique 48–bit Silicon Serial Number. If it is installed with the microprocessor, it can be read when the system is first powered up, then stored inside the Secure Microcontroller. This serializes the system. If the software ever finds a different serial number (or missing number) from the stored one, it can refuse to work. This would mean that the microprocessor had been moved.

3.11. Tamper Protection

Using a variety of tamper sensors in conjunction with the DS5002 makes the system very difficult to crack. These circuits vary from simple switches to light, temperature, pressure, or oxygen sensors. When the physical security is violated, the SDI pin is activated and the memory contents are destroyed.

3.11.1. Secure Microcontroller Features

- PiP-EC02 embedded controller platform
- AHB
- SRAM controller
- interrupt controller

- UART
- timers
- AHB – APB bridge
- external bus interface
- general purpose I/O
- 3DES cryptographic processor core
- DES – DMA controller

3.12. Strong Cryptography with Weak Microcontrollers

When a subsystem component has limited processing power and memory, advanced cryptography is usually not possible. Secure exchange of data and peer-authentication requires secrets, and microcontrollers are not very good at keeping secrets from clever hardware and software attacks. The solution is to move the cryptographic task to a device that is specially designed to perform these tasks well. This thesis will discuss the application of 1-Wire devices that perform SHA-1 hash functions to provide a low-cost, low-overhead cryptographic solution for small message encryption and authentication [29].

To keep microcontroller code space to a minimum, we will use a simple fundamental cryptographic concept called the one time pad. Given an array of bytes that comprise a message, it can be said that the byte-wise XOR (Exclusive-OR) result of that array of bytes against an array of random bytes results in an array of bytes that are equally random. In other words, no information about the message remains in the resulting array to put it another way a byte when XOR'ed against a random byte results in an equally random byte. If one was to generate an array of truly random bytes (called a pad), one could then XOR that array against any valid message and the result would be an encrypted message that is unbreakable by any cryptographic means (so long as the pad is held secret and known only to the valid participants in the conversation). This result, when XORed again against the same pad, will be restored to the original message. This is a basic tenet of cryptography the function is simple (XOR) and the power is entirely in the quality and security of the pad, not in the algorithm by which it is applied to the message.

This would seem to be a very simple, very powerful way to perform message encryption, but there are caveats:

1. The pad must be passed from the sender to the recipient in a way that it cannot be compromised. Both parties to the conversation must share the same pad.
2. If the pad is used on more than one message, its strength is greatly diminished if not wholly compromised. A new pad must be created for each message.

What is needed is the ability to generate an array of bytes (a pad) at will, and then to pass some key with the message that can be used by the recipient to regenerate the same pad. This means that each legitimate participant in the conversation must hold some secret that allows the pad to be regenerated given the key, and the secret must be protected at all costs.

In the world of cryptography, the pad generator that we have described is a function called a one-way hash. This function takes input data and generates a digest from it. This digest is entirely affected by every bit of the input data, and yet is derived in such a way that the input data cannot be discovered given the algorithm and the digest. The best-known and most secure one-way hash function is SHA-1 (secure hash algorithm).

The Dallas DS1963S SHA iButton®, and the DS2432 SHA memory chip performs all the required tasks to serve as specialized cryptographic coprocessors:

1. Inexpensive
2. Easily connected to a microcontroller using only one I/O pin
3. Can hold a secret in nonvolatile storage and protect it from attack
4. Can rotate (iterate) the secret easily and with complete security
5. Can quickly generate a cryptographically sound pad using SHA-1

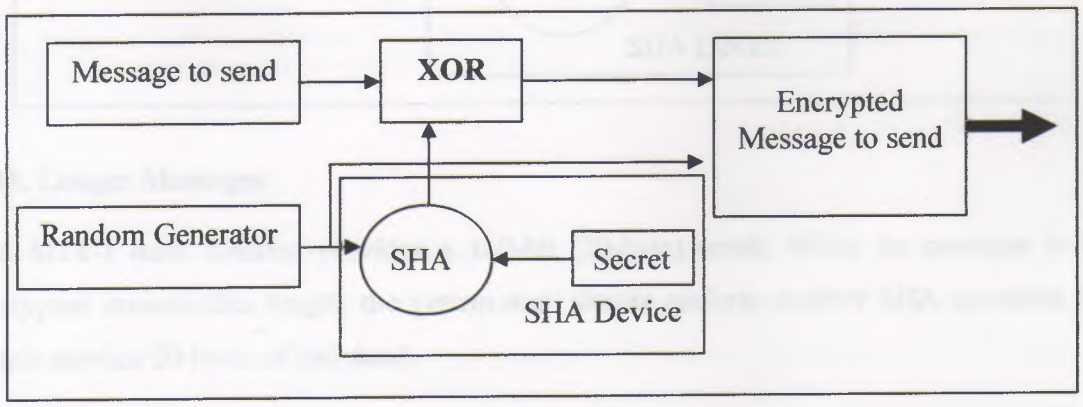
3.13. Hardware Configuration

The SHA device requires only a single microcontroller port pin and a pull up resistor. Code in the microcontroller generates the appropriate waveforms to perform two-way communication with the device at either 14KBPS or 140KBPS data rates. Each device includes a globally-unique serial number. The device has the ability to hold and protect secrets and to perform the SHA-1 hash algorithm very quickly.

3.14. The Method

To use the SHA device for peer-to-peer small message encryption, the following simple algorithm is employed [29]:

1. The microcontroller generates a random number and sends it to the device.
2. The microcontroller directs the device to generate a SHA-1 digest using the random number and the secret.
3. The microcontroller reads the 160-bit digest from the device.
4. The microcontroller XORs each byte of the message with a byte of the digest (the pad) to obtain the encrypted message.
5. The microcontroller concatenates the random number and the encrypted message and transmits the result to the peer.

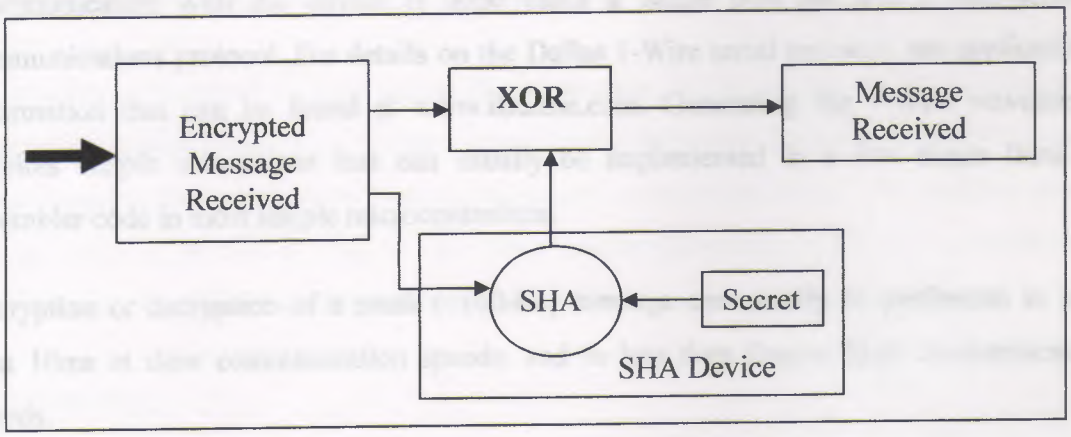


When an encrypted message arrives, the following algorithm is employed:

1. The random number portion of the message is sent to the device.
2. The microcontroller directs the device to generate a SHA-1 digest using the random number and the secret.
3. The microcontroller reads the 160-bit digest from the device.
4. The microcontroller XORs each byte of the message with a byte of the digest (the pad) to obtain the original message.
5. The microcontroller processes the decrypted message.

Despite the simple appearance of this algorithm, it is quite secure. The microcontroller needs only perform basic 1-Wire communications with the device and then XOR the SHA-

1 digest, byte for byte, against the message data. Security is assured by the strength of the SHA-1 function. Because the SHA-1 hash function is not reversible, the secret cannot be derived from the message traffic. Without the secret, there is no way to decipher or falsify a message. The random seed value used with each message makes every message unique, and makes the deciphering messages all but impossible.



3.15. Longer Messages

The SHA-1 hash function provides a 160-bit (20-byte) result. When the message to be encrypted exceeds this length, the system may simply perform another SHA operation (to obtain another 20 bytes of pad data).

3.15.1. Replay Attacks

This encryption scheme provides authentication of the source of each message (because only a valid system participant could have generated a valid message) as well as message data security. However, replay attacks (where a previous valid message is captured and sent again at a later time) are still possible. Several simple methods can protect against this. One method is to include a counter in each message that increments when the message is sent, and then program the recipients to reject duplicate messages.

Another method is to have the recipient send a random number (called a challenge) that the sender then includes in the encrypted message. Because the random challenge is very likely not the same for any two messages, a replayed message will be rejected.

3.15.2. Random Numbers

The system described herein relies on the cryptographic quality of the random numbers generated by the microcontrollers.

3.15.2.1. 1-Wire Serial Communication

Communication with the device is done using a single port pin and a well-defined communications protocol. For details on the Dallas 1-Wire serial protocol, see applications information that can be found at www.iButton.com. Generating the 1-Wire waveforms requires simple subroutines that can usually be implemented in a few dozen lines of assembler code in most simple microcontrollers.

Encryption or decryption of a small (<160-bit) message can usually be performed in less than 10ms at slow communication speeds, and in less than 2ms at high communication speeds.

3.15.3. The SHA Device

There are two forms of SHA device available, each with different characteristics. The DS1963S is a nonvolatile memory with an SHA-1 engine, plus a lithium power source, inside a stainless steel container called an iButton. This device has sixteen 32-byte pages of memory and eight separate SHA-1 secrets. The DS1963S is used in a coprocessor mode to perform the type of SHA-1 function required for small message encryption.

The DS2432 is a lower-cost, surface mounted EEPROM version of the DS1963S in a very small TSOP-8 package. It has four 32-byte pages of memory and a single SHA-1 secret. Being surface-mounted, battery-less and lower in cost than the DS1963S, the DS2432 is perhaps better suited for this type of application, but current versions of the DS2432 do not provide the coprocessor mode required for this functionality. Future versions of the DS2432 will include this option.

3.15.4. Secret Rotation

The SHA devices also provide internal mechanisms to perform secret rotation. The system may send a rotation message to the device and ask that the message be hashed against the old secret to generate a new secret. The device microcontroller does not need to know the old secret to generate the new secret, and the new secret is never revealed never visible outside the device. In this manner, the system-wide secret can be easily changed (rotated).

An attacker is required to have access to the original secret and the rotation message to figure out the new secret. This allows a system to rotate secrets from time to time to assure secret security. This mechanism also allows system secrets to be installed in multiple pieces and by different parties so that no one party knows everything required to generate the system secret. This secret sharing method further enhances system security.

3.16. Summary

Although there is a lot of Microcontroller cryptography techniques exist we can implement all but with the time we have considered above mentioned techniques and after analyzing these we have adopted one technique to make it running as an implementation part.

3DES security is just the kind of value-add customers are turning to as a way of differentiating their products from the competition. By using a co-development approach instead of coding the design first and then synthesizing it, we were able to design in security in two modules.

The DS1963S SHA iButton can be used with small microcontrollers to provide strong encryption and authentication of control and status messages, telemetry, or sensitive process control information. For low cost and low overhead, it provides nonvolatile memory, secure secret storage, secret sharing and rotation, fast SHA-1 pad generation, and a globally unique serial number. A simple microcontroller needs only provide a single port pin and a few dozen lines of code to attain quality cryptographic security. Future versions of the DS2432 will lower the cost and space requirements even further in an EEPROM type device.

4. INTERFACING & MICROCONTROLLERS

4.1. Overview.

Microprocessor interfacing technology is evolving as dynamically as the VLSI components it supports. The techniques of interfacing are based on fundamental electrical engineering principles, advanced micro fabrication rules, and computer hardware and software concepts. It is the solemn aim of this thesis to bind together these different ideas and develop basic skills of interfacing. The thesis sets off with the intricate aspects of interfacing the parallel port using the IEEE488 protocols. Then it delves into serial ports and the RS232 protocols which govern the abyss of serial communication.

In the past 20 years, there has been little substantial change in the actual sequence of steps we go through when designing a system. On the other hand, there has been a *significant* change in emphasis of the design steps: as the later phases have become more or less automated, designers have come to focus more and more on the earlier, more abstract phases of the system-design process. This shift in focus has enabled designers to create increasingly complex systems in shorter lengths of time. In designing such complex systems, achieving correct functionality is far more important and more difficult than rninimizing silicon area or program-memory size. The system functionality, however, can best be understood during the earlier design steps, before a lot of implementation details have been added, and this is why these early phases of the process have become so crucial in system design.

The market for products or services that are Internet related is HOT! Increased amounts of money and design resources are being thrown at these products and services. One significant portion of this trend is to embed the Internet or, in other words, make embedded products have the capability to connect to the Internet. It is estimated that by the year 2005, the number of embedded applications with the ability to connect to the Internet will be larger than the number of PCs by a factor of 100 or more. The latest in PDAs and cellular telephones allow the user to access stock quotes, sports scores, Email, and more. The other advantage of the movement to embed the Internet is

enabling client devices, such as appliances and vending machines, to connect to the Internet and upload status information.

4.2. Interfacing the Parallel Port

The Parallel Port is the most commonly used port for interfacing home made projects. This port will allow the input of up to 9 bits or the output of 12 bits at any one given time, thus requiring minimal external circuitry to implement many simpler tasks. The port is composed of 4 control lines, 5 status lines and 8 data lines. It's found commonly on the back of your PC as a D-Type 25 Pin female connector. There may also be a D-Type 25 pin male connector. This will be a serial RS-232 port and thus, is a totally incompatible port.

Newer Parallel Port's are standardized under the IEEE 1284 standard first released in 1994. This standard defines 5 modes of operation which are as follows, Compatibility Mode, Nibble Mode, Byte Mode, EPP Mode *(Enhanced Parallel Port)*. ECP Mode *(Extended Capabilities Port)*.

The aim was to design new drivers and devices which were compatible with each other and also backwards compatible with the Standard Parallel Port (SPP). Compatibility, Nibble & Byte modes use just the standard hardware available on the original Parallel Port cards while EPP & ECP modes require additional hardware which can run at faster speeds, while still being downwards compatible with the Standard Parallel Port. Compatibility mode or "Centronics Mode" as it is commonly known, can only send data in the

1. Forward direction at a typical speed of 50 Kbytes per second but can be as high as 150+ bytes a second. In order to receive data, you must change the mode to either Nibble or Byte mode. Nibble mode can input a nibble *(4 bits)* in the reverse direction. E.g. from device to computer. Byte mode uses the Parallel's bi-directional feature *(found only on some cards)* to input a byte *(8 bits)* of data in the reverse direction. Extended and Enhanced Parallel Ports use additional hardware to generate and manage handshaking.

This limits the speed at which the port can run at. The EPP & ECP ports get around this by letting the hardware check to see if the printer is busy and generate a strobe and /or

appropriate handshaking. This means only one I/O instruction need to be performed, thus increasing the speed. These ports can output at around 1-2 megabytes per second. The ECP port also has the advantage of using DMA channels and FIFO buffers, thus data can be shifted around without using I/O instructions.

4.3. Introduction to Universal Serial Bus:

In October, universal serial bus got a boost when the seven leading vendors of the USB 2.0 Promoter Group-Compaq, Hewlett-Packard, Intel, Lucent Technologies, Microsoft, NEC, and Philips-said that the upcoming USB 2.0 specification would be 40 times faster than USB 1.1. Higher speeds will fuel quick growth for USB, experts predict. The USB 2.0 spec-which will be finalized during the first quarter of 2000, with products due in the second half-is expected to have a throughput speed of 480 Mbps. It can also adapt 127 peripherals on the port and peripherals can be connected without having to restart the IBM PC.

Thanks to another USB feature known as "hot-swapping" you don't even need to shut down and restart your PC to attach or remove a peripheral. Just plug it in and go! The PC automatically detects the peripheral and configures the necessary software. This feature is especially useful for users of multi-player games, as well as business and notebook PC users who want to share peripherals.

USB also lets you connect many peripherals at one time. Many USB PCs come with two USB ports. And special USB peripherals -- called USB hubs -- have additional ports that let you "daisychain" multiple devices together.

An external universal serial bus port eliminates the need for PCs to have multiple ports-serial and parallel, mouse, monitor, and keyboard. It can be used to add multiple peripherals-printers, scanners, desktop backup devices, networking devices, and modems-all without requiring users or integrators to open the PC's chassis. The changes being made to USB are enabling it to surpass the IEEE 1394 high-speed interface standard, known as FireWire, which runs at about 400 Mbps

4.4. Interfacing the Serial Port.

The Serial Port is harder to interface than the Parallel Port. In most cases, any device you connect to the serial port will need the serial transmission converted back to parallel so that it can be used. This can be done using a UART. On the software side of things, there are many more registers that you have to attend to than on a Standard Parallel Port (SPP).

The advantages of using serial data transfer rather than parallel are:

1. Serial cables can be longer than Parallel cables with a maximum voltage swing of 50V (± 3 to ± 25) as compared to parallel, 5V (0-5V). Hence cable loss is not a problem for serial cables.
2. Wires required are less compared to parallel transmission. Null modems taking only 3 core cables for long distance transmission as against 19-25 core cables required by parallel.
3. Microcontrollers have also proven to be quite popular recently. Many of these have in built SCI (Serial Communications Interfaces) which can be used to talk to the outside world. Serial Communication reduces the pin count of these MPU's.
4. For Infra red devices using Serial communication, deciphering the bits is much easier than in case of Parallel transmission.

The UART 8250 and USART 8251 are designed specifically for this purpose and they use the following reference standards [1].

4.5. RS 232 C

This EIA standard introduced in 1962 defines the interface between data terminal equipment and data communication equipment employing serial binary data interchange (both synchronous and asynchronous). This standard describes functions of 25 signal and handshake pins for serial data transfer. It also describes voltage levels, impedance levels, rise and fall times, maximum bit rate and maximum capacitance for these signal lines. It states that the DTE connector should be male and the DCE connector should be female. The voltage levels 3 to 15 volts define logic 0 and –3 to –15 volts define logic 1 .The control and timing signals are compatible with TTL level. Since data lines are

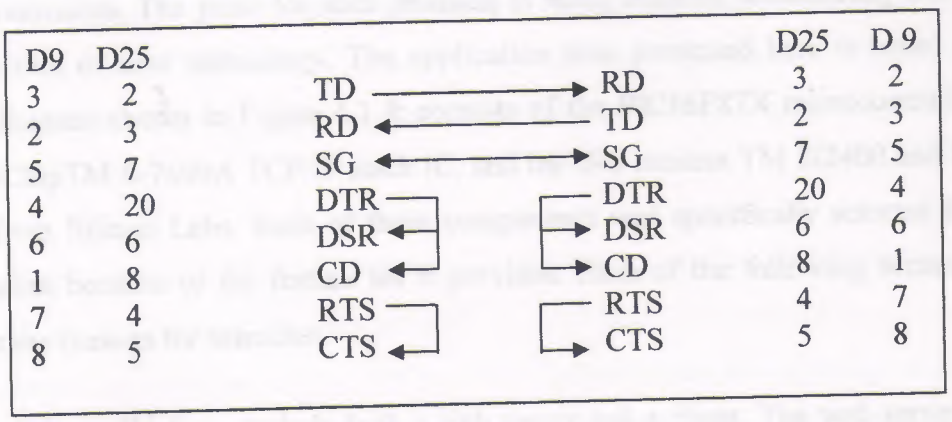
incompatible with TTL logic, voltage translators called *line drivers* and *line receivers* are required to interface TTL logic with RS 232 signals. Refer to figure below.

The standard specifies that "the driver shall be able to withstand and open circuit, a short circuit between the conductors carrying that interchanged circuit in the interconnecting cable or any passive non inductive load connected between that interchanged circuit and any other interchanged circuit including signal ground, without sustaining damage to itself or its associated equipment". In an RS-232C application an RS-232C interface is connected between DTE and DCE. The CCITT (International Telegraph and Telephone Consultative Committee) Standard V.24 is identical to RS-232C.

4.5.1. Null Modems

A Null Modem is used to connect two DTE's together. This is commonly used as a cheap way to network games or to transfer files between computers using Zmodem Protocol, Xmodem Protocol etc. This can also be used with many Microprocessor Development Systems.

It only requires 3 wires (TD, RD & SG) to be wired straight through thus is more cost effective to use with long cable runs. The aim is to make to computer think it is talking to a modem rather than another computer. Any data transmitted from the first computer must be received by the second thus TD is connected to RD. The second computer must have the same set-up thus RD is connected to TD. Signal Ground (SG) must also be connected so both grounds are common to each computer.



The Data Terminal Ready is looped back to Data Set Ready and Carrier Detect on both computers. When the Data Terminal Ready is asserted active, then the Data Set Ready and Carrier Detect immediately become active. At this point the computer thinks the Virtual Modem to which it is connected is ready and has detected the carrier of the other modem.

All left to worry about now is the Request to send and Clear to Send. As both computers communicate together at the same speed, flow control is not needed thus these two lines are also linked together on each computer. When the computer wishes to send data, it asserts the Request to send high and as it's hooked together with the Clear to Send, It immediately gets a reply that it is ok to send and does so.

Notice that the ring indicator is not connected to anything of each end. This line is only used to tell the computer that there is a ringing signal on the phone line. As we don't have a modem connected to the phone line this is left disconnected.

4.6. Embedding PIC micro® Microcontrollers in the Internet

The term microcomputer is used to describe a system that includes a microprocessor, program memory, data memory, and an input/output (I/O); some microcomputer systems include additional components such as timers, counters, analogue-to-digital converters and so on. Thus, a microcomputer; system can be anything from a large computer system having hard disks, floppy disks and printers, to single chip computer Systems. In this thesis we are going to consider only the type of microcomputers that consist of a single silicon chip. Such microcomputer systems are also called microcontrollers. The price for such products is being reduced dramatically due to the introduction of new technology. The application note presented here is based on the block diagram shown in Figure 4.1 It consists of the PIC16F87X microcontroller, the Seiko iChipTM S-7600A TCP/IP stack IC, and the ISO modem TM Si2400 and Si3015 DAA from Silicon Labs. Each of these components was specifically selected for this application because of the feature set it provides. Each of the following sections will present the reasons for selection.

The sample applications include both a web server and a client. The web server stores the HTML page in an external serial EEPROM and dynamically inserts temperature and

potentiometer settings into the HTML file as it is being sent. The client application mimics a vending machine that uploads status information at predefined intervals.

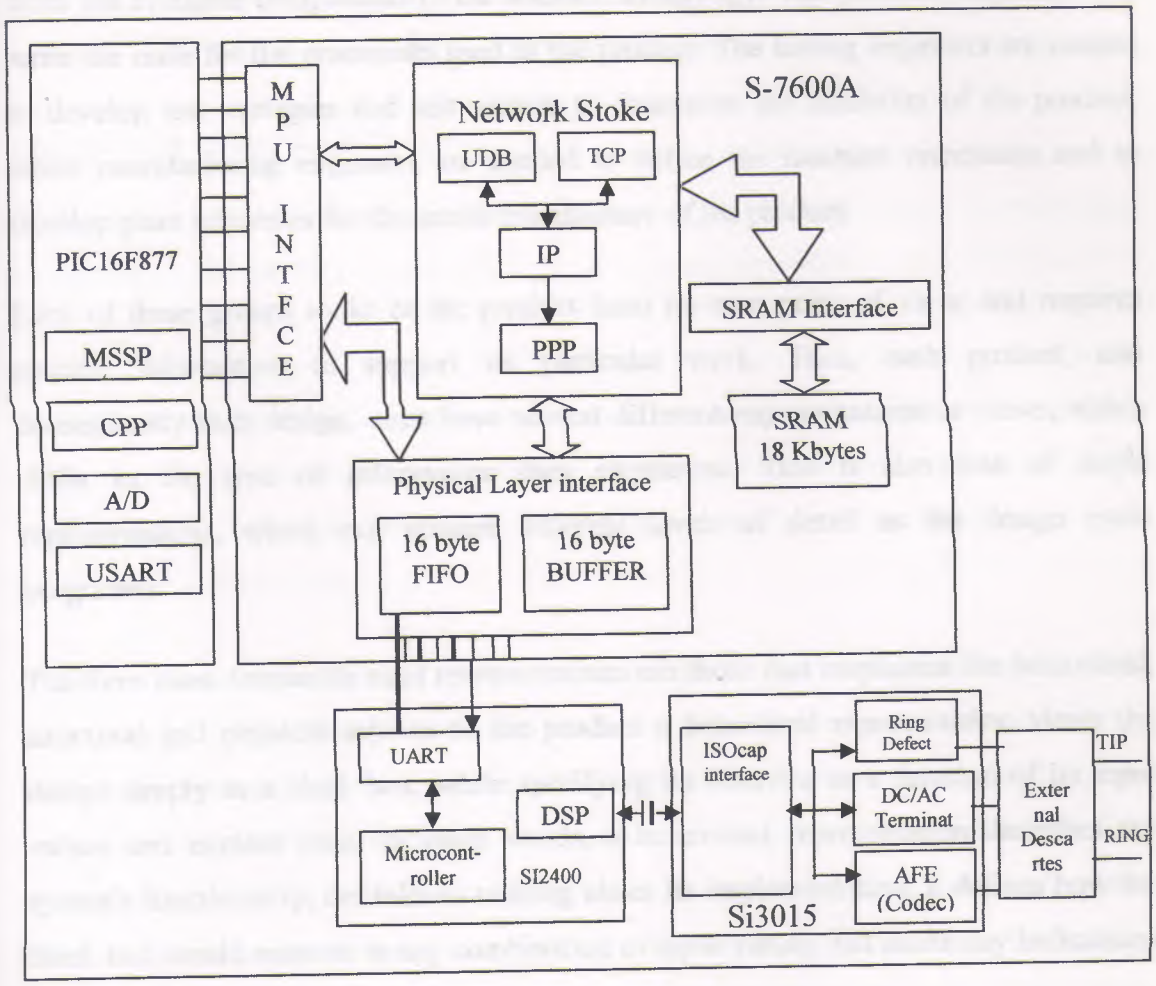


Figure 4.1 Block Diagram

4.7. Design Representation

For any particular product, the design process will always start with conceptualizing the product's functions, and will end only when we have produced a manufacturing blueprint. Before it is finished, many different people will have been involved in this process.

For example, the marketing department is needed to study market needs and to determine requirements for the new product. A chief architect is needed to convert those requirements into architecture for the product. Technologists are involved in selecting the technology, the possible components and the suppliers, while computer-aided-design and computer-aided-software-engineering groups must either acquire or develop

the tools necessary to support the design of the product, including each of its parts. A design team will develop the blueprint that indicates how to manufacture the product from the available components in the selected technology. The software engineers will write the code for the processors used in the product. The testing engineers are needed to develop test strategies and test vectors to determine the reliability of the product, while manufacturing engineers are needed to define the machine operations and to develop plant schedules for the actual manufacture of the product.

Each of these groups looks at the product from its own point of view, and requires specific information to support its particular work. Thus, each product, and consequently each design, must have several different representations or views, which differ in the type of information they emphasize. This is also true of single representations, which can acquire different levels of detail as the design cycle progresses.

The three most frequently used representations are those that emphasize the behavioral, structural and physical aspects of the product a behavioral representation views the design simply as a black box, while specifying its behavior as a function of its input values and expired time. *In other words, a behavioral representation* describes the system's functionality, but tells us nothing about its implementation; it defines how the black box would respond to any combination of input values, but omits any indications about how we would design that box.

A structural representation, by contrast, begins to answer some of these questions, as it serves to define the black box in terms of a set of components and their connections. In other words, this representation focuses on specifying the product's implementation, and even though the functionality of the black box can be derived from its inter-connected components, the structural representation does not describe the functionality explicitly.

A physical representation carries the implementation of the design one step further, specifying the physical characteristics of the components described in the structural representation. For instance, a physical representation would provide the dimensions and location of each component, as well as the physical characteristics of the

connections between them. Thus, while the structural representation provides the design's connectivity, the physical representation describes the spatial relationships among these interconnected components, describing the weight, size, heat dissipation, power consumption and position of each input or output pin in the manufactured design.

4.8. Microcontroller Architecture

The simplest microcontroller architecture consists of a microprocessor memory, and input/output. The microprocessor consists of a central processing unit (CPU) and the control unit (CU).

The CPU is the brain of a microprocessor and is where all of the arithmetic an (logical operations are performed. The control unit controls the internal operations of the microprocessor and sends control signals to other parts 0 the microprocessor to carry out the required instructions.

Memory is an important part of a microcomputer system. Depending upon the application we can classify memories into two groups: program memory and data memory. Program memory stores all the program code. This memory is usually a read-only memory (ROM). Other types of memories, e.g. EPROM and PEROM flash memories are used for low-volume applications and also during program development. Data memory is a read/write memory (RAM). In complex applications where there may be need for large amounts of memory it is possible to interface external memory chips to most microcontrollers.

4.9. Microcontroller

The microcontroller is the primary component of the application, not necessarily a part of the Internet communications. In some cases, it may be advantageous to offload the Internet communications to an external device, so the microcontroller can focus on the details of the application, as well as the application layer of the Internet Protocol Stack (see Figure 4.2).

One feature that provides the most flexibility for an embedded application is FLASH-based program memory.

TABLE 4.1 PIC16F87X PRODUCT FAMILY

Features	PIC16F870	PIC16F871	PIC16F872	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz						
Resets	Power-on Reset, Brown-out Reset, Power-up Timer, Oscillator Start-up Timer						
FLASH Program Memory	2K	2K	4K	4K	8K	8K	8K
Data Memory	128	128	128	192	192	368	368
EEPROM Data Memory	64	64	64	128	128	256	256
Interrupts	10	10	11	13	14	13	14
I/O Pins	22	33	22	22	33	22	33
Timers	3	3	3	3	3	3	3
Capture/Compare/ PWM	1	1	1	2	2	2	2
Serial Communication	USART	USART	MSSP	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communication		PSP			PSP		PSP
10-bit A/D Channels	5	8	5	5	8	5	8
Instruction Set	35 Instructions						
Packages	Packages 28-pin DIP, SOIC, SSOP	40-pin DIP 44-pin PLCC, TQFP	28-pin DIP, SOIC, SSOP	28-pin DIP, SOIC	40-pin DIP 44-pin PLCC, TQFP	28-pin DIP SOIC	40-pin DIP 44-pin PLCC, TQFP

Information that is updated frequently can be stored in the data EEPROM, which has increased endurance over the FLASH program memory. This data can be modified without interrupting the operation of the application. Other information that is rarely changed can be stored in FLASH program memory. When updating these locations, the microcontroller ceases to execute instructions, but continues to clock the peripherals and queue interrupts. The programming operation is self-timed, i.e., an internal clock source applies the high voltage to the memory cells to erase and/or write the data for a predetermined amount of time. This frees the designer from the task of timing the event.

It also reduces the risk of the high voltage being applied too long, which damages or reduces the endurance of the cell, or not longs enough, which does not fully program the memory location.

4.10. TCP/IP STACK & Embedded Microchip

The S-7600A TCP/IP Stack IC from Seiko Instruments was designed specifically for this type of market. It integrates the TCP/IP Stack engine, 10 Kbytes of RAM, microcontroller interface and UART into a single chip. Once configured, it acts like a data buffer. Data to be transmitted, up to 1024 bytes, is stored in the internal RAM buffer and the TCP/IP engine appends the various headers and checksums. It then transmits this packet from the UART. When packets are received, the TCP/IP engine determines if the IP address and port number match those set during configuration, calculates and verifies the checksums, and transfers the data contents of the packet to a buffer. It then uses interrupt lines to indicate there is data available to the microcontroller.

The complete lists of features for the S-7600A are:

- Implements PPP, IP, TCP, UDP and PAP.
- Two general purpose sockets.
- Two parallel interfaces (68K/x80 Motorola/Intel MPU bus) or synchronous serial interface.
- On-chip UART physical layer transport interface.
- 256 kHz typical operating frequency.
- Low power consumption (0.9 mA typical, 1.0 μA standby).
- 2.4 V to 3.6 V operating voltage range.

The designer now has full Internet capabilities without any of the limitations of the software implementations. The bulk of the program memory on the microcontroller can now be used for the main application and also for implementing some of the Application Layer protocols previously described. The size of the program memory is now dependent on the application and can be scaled accordingly. The S-7600A delivers Internet capability to the bulk of microcontrollers that were previously constrained due to program and data memory size.

4.11. Modem

The last key ingredient is the communications medium used to connect to the Internet. The type of medium selected is highly application dependent but can include:

- Wireless
- Cellular
- Power Line Carrier
- POTS (Standard telephone lines)

All of these mediums require some infrastructure to be in place before the embedded device can communicate. Both wireless and cellular transceivers require antennas to be placed in the surrounding area to provide the communication channel. While most areas have some of this infrastructure in place, there are areas that are not completely covered. Everyone has probably experienced this with a cellular telephone at one time or another. Power line carrier also requires infrastructure to be in place. There has to be some sort of transceiver at the other end of the power lines that communicates with the embedded application. This infrastructure for this technology does not currently have the widespread use that wireless or cellular offer and therefore, the costs to build this infrastructure would be substantial. The standard telephone lines are everywhere.

The telephone poles, wiring, relay stations, etc., are already in place. The cost of building the infrastructure is zero and therefore, makes the most sense for the bulk of embedded applications. There will be applications where the other mediums are needed, but the application will be able to justify and therefore, absorb the additional costs associated with using the respective mediums. Usually, the telephone modem technology is significantly less expensive than that of other mediums. It also fits better into embedded applications due to size and power consumption. The key to modem selection is to find one that is highly integrated, i.e., smaller in size. This is important due to the fact that most embedded applications are small. This modem should be easy to use and provide all the necessary features in a single package. Fortunately, the folks at Silicon Laboratories have developed an embedded modem that may be one of the best designs ever to hit the streets. The first feature that stands out is the size. Figure 4.3 shows the size of the Silicon Labs design compared to a standard modem design. NO relays. NO opt isolators. NO transformers. This design has the Si2400 modem and the

Si3015 DAA chip and some passive devices (resistors, capacitors, diodes and transistors). The secret is in the ISO cap TM Technology, used to transfer control and data over a capacitive interface as shown in Figure 4.4 This interface provides a low cost solution based off standard high voltage capacitors and over 2400 V of isolation.

Never before has the embedded control electronics industry seen a modem design this integrated AND this small AND this CHEAP! But one question remains;

Why 2400 baud? Isn't that baud rate a little slow to use for Internet applications, even embedded Internet applications?

The answer is quite simple. If the application was a web server, then yes, a 2400 baud modem is not practical. But it was already established that a web server was not practical for the embedded world. A typical embedded application will only transfer several hundred bytes of data. When looking at the complete connect and transfer time of one session, a 2400 baud modem will connect in approximately three seconds and upload 200 bytes of data in 0.833 seconds (200 bytes x 10 bits/byte x 1s/2400 bits) for a total of 3.833 seconds. A 56K modem will connect in approximately 15 seconds and transfer 200 bytes in 0.036 seconds (200 bytes x 10 bits/byte x 1s/56000 bits). This calculation shows that a 2400 baud modem can connect to the ISP, dump the data to the server and disconnect before the 56K modem even establishes a connection to the modem on the other end of the line. It just doesn't make sense, especially when you consider the price of the 2400 baud modem versus the 56K modem. Another feature of a telephone-based system is choosing the ISP to make the Internet connection.

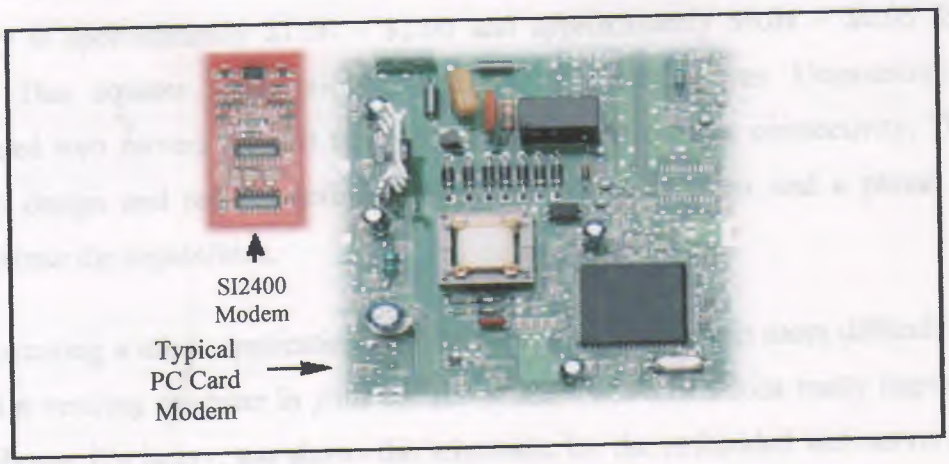


Figure 4.3 Modem Size Comparisons

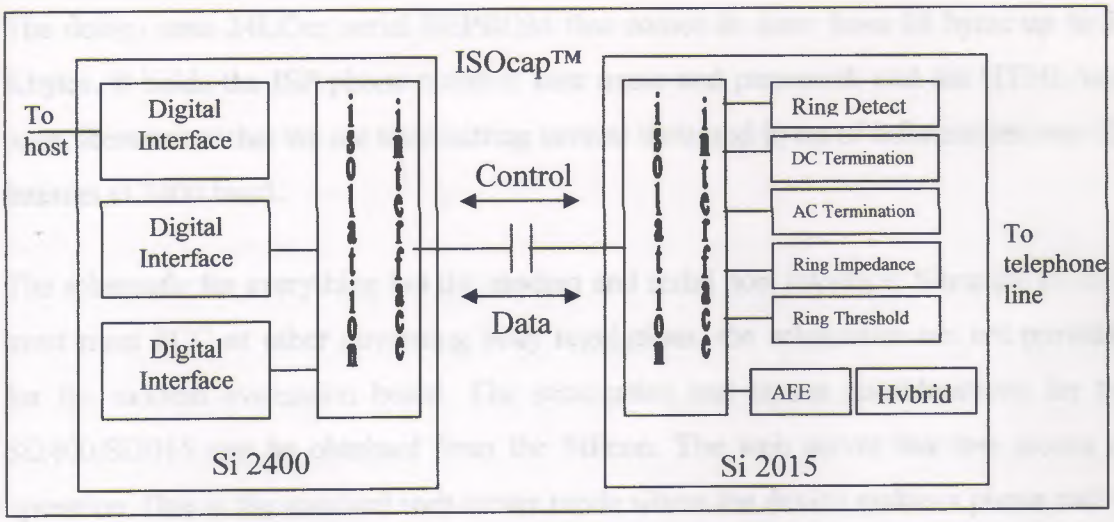


Figure 4.4 ISOcap™ Interface

Everyone hears about the high speed Internet links such as cable modems. Most providers are targeting customers that want high-speed access for web browsing. According to the estimates, this market which itself is very large, will be dwarfed by the embedded devices. Some companies are starting to realize this fact and are catering towards these embedded applications with low speed modems.

4.12. Web Server Application

The embedded web server application is more for show and tell. As mentioned before, it is not really a practical use of the hardware. The memory sizes required to serve web pages and data files far outweighs that which can be found on a typical microcontroller. In fact, if the price of non-volatile semiconductor memory and that of hard drives were compared, the results would show that the average price per megabyte of FLASH memory is approximately $1.00 – $2.00 and approximately $0.01 – $0.05 for hard drives. That equates to a ratio of 40:1 favoring hard drives. Demonstrations of embedded web servers are just that, demonstrations of Internet connectivity. They are easy to design and require nothing more than a web browser and a phone line to demonstrate the capabilities.

Demonstrating a client application such as a vending machine is more difficult. Toting around a vending machine in your car for product demonstrations really impacts your gas mileage. It's heavy, too shows the schematic for the embedded web server. It uses the PIC16F877, the S-7600A TCP/IP stack IC, the Si2400 modem and Si3015 DAA.

The design uses 24LCxx serial EEPROM that comes in sizes from 16 bytes up to 32 Kbytes. It holds the ISP phone number, user name and password, and the HTML web page. Remember that we are transmitting several thousand bytes of information over the Internet at 2400 baud.

The schematic for everything but the modem and serial port interface. Since the modem must meet FCC or other governing body regulations, the schematics are not provided for the modem evaluation board. The schematics and layout considerations for the Si2400/Si3015 can be obtained from the Silicon. The web server has two modes of operation. One is the standard web server mode where the device makes a phone call to the local ISP and establishes an IP address.

The user may access the web page by typing in the IP address displayed on the LCD display into any web browser. In the web page some variable information, such as number of hits, temperature where the web server is, IP address, and a potentiometer setting. This information is dynamically inserted into the web page as it is transmitted. The other mode is a configuration mode, which allows the ISP information and web page to be downloaded into the serial EEPROM. The ISP information includes the phone number, user name and password. The size of the serial EEPROM is application dependent. It can range from 16 bytes (24LC00) to 32 Kbytes (24LC256). The board is currently using a 16 Kbytes device, the 24LC64. To configure the web server, the RS-232 interface on the modem evaluation board is connected to the USART module on the PIC16F877. Ring Detect DC Termination Ring Impedance AC Termination Ring Threshold AFE Hybrid Isolation Interface Isolation Interface to Host Digital Interface Micro-Controller DSP **Si2400 Si3015 to Telephone Line Data Control ISOcap™.**

Any terminal program will work, as the PIC16F877 displays a text menu that allows the user several options:

1. Enter user name
2. Enter password
3. Enter phone number
4. Download HTML file
5. Or exit configuration mode

Each piece of information has specific memory locations reserved in the serial EEPROM. 32 bytes are reserved for both the user name and password. The ISP phone number takes an additional 16 bytes. Finally, 32688 bytes are available to store the HTML file. The serial interface must use hardware flow control, otherwise the USART buffer will be overrun, due to the programming time required by the serial EEPROM. The application provides lots of status information. Messages are displayed when a call is being made, a dial tone is detected, and a ring occurs, a busy phone line is detected, or when the modem finally makes a connection. The display will show the IP address once the modem has made the connection. There are also a couple of LEDs that indicate the status of the web server. The first LED shows if the modem is connected. The second LED flickers when the web server is being accessed. Special characters are used to allow the web server to insert variable data into the web page. The following information can be displayed when these characters are found in the HTML web page:

- %a displays the IP address for web page reloads function.
- %c inserts the current temperature in degrees Celsius.
- %f inserts the current temperature in degrees Fahrenheit.
- %h displays the number of times the web site has been accessed.
- %p inserts the current value of the potentiometer several modifications had to be made to the modem evaluation board.

1. It was originally designed to interface the serial port on the PC directly to the modem. Using a terminal program, the user could make or receive phone calls. This interface was hacked to allow stacking of the control board on top of the modem evaluation board..

4.13. Client Application

This application represents a typical embedded Internet application, where the embedded device is the client and is capable of connecting to a server to upload information and download new information, or firmware. In this case, a vending machine that receives new information. The vending machine application has an LCD display that shows the current items in the machine and the price for each. It will "dispense" items until the machine is empty. At any time, a push-button switch may be

pressed to start the connection to the server via the Internet. The modem dials an ISP, makes a connection to the server, and receives new names and prices of items to be dispensed. The hardware design is a subset of the web server application.

This design removes the serial EEPROM, potentiometer and temperature IC. It adds some push-button switches for the additional user interface required. It uses the same modem evaluation board as the web server with all the same modifications. The vending machine has two modes of operation. It has the standard operating mode of reading the push-button switches and "dispensing" items based on which button is pressed. It tracks the number of items remaining in the vending machine and the total amount of money collected.

The second mode of operation is the Internet connection. Most of the code to interface with the S-7600A is the same as the web server, with the exception that it is now a TCP client instead of a TCP web server. It must also know what the IP address and port number of the server is before it can make a connection to the server. This means that more than a web browser is required to complete the connection on the server side. There must be a program running on the server that listens to a port. Once connected, the transfer of information may take place between the client and the server. In this application, the Internet connection provides the names and prices of new items. Every connection to the server downloads two new item names and prices that are then programmed into data EEPROM. Since these are values that could change frequently, the data EEPROM was used for nonvolatile storage of the information, due to the higher endurance. The same methods presented here can be used in conjunction with the boot loader of AN732 to the new source code into FLASH program memory. The data that is transferred between the client and the server has some handshaking built in. Once the connection is established, the client waits for a response from the server. The value of the data is not important, only the response from the server, so the buffer is emptied without any processing of the information. The client now responds with an index number between 0 and 9. This index is used by the server to extract the next vending machine item names and prices out of a database. The format of this data is ~ ~ # ~ ~ where # is the index value 0-9. The server will then respond with the new names and prices in the following format: ~ <name1>; <name2>; <price1>; <price2>;

The tilde character is used to denote the start of the string. Each of the names and prices is a null terminated ASCII string and they are delimited using semicolons. Once the client receives this information and updates the data EEPROM, the connection with the server is terminated. At this point, the client must be reset through a MCLR Reset, or by cycling the power. It now switches back to the normal mode of operation, using the new names and prices provided by the web server. Other information, such as total amount of money collected and the number of remaining items, could have been transmitted back to the server, but the application was kept simple for both the client and the server. This interaction is highly application dependent and can be easily adapted based on the system requirements.

4.14. Summary

The matters covered and the topics discussed in the thesis are just drops in the ocean of the interfacing world. This thesis attempts to bind together the different interfacing techniques which are implemented for different applications in the computer world. A general coverage of the basic interfacing techniques is done and recent developments like the EPP, ECP and the USB are discussed. Most of these techniques are used for communication between PCs and for Data Acquisition Systems. Though the age old methods of interfacing are cast in the limbo of oblivion, this thesis makes a modest attempt to relate its use in the rising technologies that adorn the dawn of a new era.

The move to embed the Internet is creating many new and fascinating devices for all different types of markets. Cellular phones and PDAs are the latest devices to add Internet capability. Soon many household appliances, such as refrigerators, will have Internet capability and these embedded applications will dominate the Internet. These devices can Internet-enable any application that has already used most of the available microcontroller resources to control the application. In most cases, a microcontroller cannot afford to dedicate 5 Kbytes of program memory and a significant portion of data memory for Internet connections. This need has created devices such as the S-7600A and Si2400 specifically for the embedded Internet markets.

5. CHOOSING A SECURITY ALGORITHM

5.1 Overview

The aim of this thesis is to implement and encryption algorithm which can be used by an embedded microcontroller system. One of the major problems here is that the microcontrollers have very limited resources. Most microcontrollers are equipped with only a few k-bytes of program memories and only several hundred bytes of data memories for example; one of the most popular microcontrollers PIC16F84 has only 1k-byts of program memory and 68 bytes of RAM data memory. The maximum clock frequency of this microcontroller is only 4MHZ. With these revere constraints its not on task to implement advanced security algorithm on such devices.

This chapter gives and overview of the popular and small-size encryption algorithms and gives recommendations for making a selection. The algorithms are simulated on PC in order to access their suitability.

5.2. Security Requirement in Embedded Applications

There are many embedded microcontroller applications which require different reveal of security. Some applications may require no security at all, while some other may require top security.

In general, we can classify the security requirement of embedded microcontroller applications as follows.

- **No Security:** these are embedded microcontroller applications where are no security requirements. Data can be sending to the microcontroller as raw plain text. For example, the microcontroller used in standard home TV remote controller applications, do not require any security.
- **Low level security:** these are embedded systems where some level of security is required. For example the microcontroller used in office automation products (e.g. a printer) may require low level of security.
- **High level security:** some embedded system such as controlled entry and exit systems usually require higher security levels. The encryption algorithm chosen for such systems should be more difficult to decode by unauthorized persons.

- **Top levels security:** Most embedded systems in military applications usually require top levels of security. These systems should have high resistance to unauthorized attacker. A compute encryption algorithm is usually chosen for such applications.

5.3. Cryptography Basics

When privately communicating with another patron, the use of a secret code can privet unintended recipients from reading the messages. Both the sender and receiver must agree on a system to the used. If the repaint receivers an encoded message without knowing how to decode the message, the message is useless. The message to the encoded is usually called the "plain text", and the encoded reaction of the message is called the "cipher text".

Cryptography is divided into two parts "private-key encryption" and "public-key encryption". A private key cryptosystem is one that s intended to the used awing a small group of regal and this type of cryptosystem are easily controlled. A public-key encryption is at a much loge scale. There are usually many participants as well as many attempting to break the codes. This cryptosystem is not security, is usually large and requires sophisticated loads.

Some commonly used private-key encryption algorithms are [2].

- Caesar Cipher
- Affine Cipher
- Random Cipher
- Vigenere Cipher
- Transposition Cipher
- Digital Encryption standard (DES)

As we will see later on, most of the above algorithms are simple do not require large resources.

Some commonly used public-key encryption algorithms are [2].

- Rivest, Shamir, Adelman (RSA)
- Pretty Good Privacy (PGP)

All encryption algorithms are based on some kind of key which is used to decode the plain text. This key can be a simple number or a complex algorithm. The choice of the key affects the security level and the complexity of the encryption algorithm for examples Caesar Cipher algorithm a simple shift of the letters is used a key. In more complex algorithms such as the RSA, Pair of keys (usually very large numbers), are public, and are private are used.

Because the embedded microcontroller systems have very limited resources, it is not possible to implement complex algorithms or then which may require large amount of program or data memories. Processing powers of the microcontroller can also be a problem when complex encryption algorithms are implemented.

In this thesis we are only interested in simple, small-size, but effective encryption algorithms. i.e. Algorithm which can be implemented with less than 1k-byte of program memory and few MHZ chick speed.

The encryption algorithms studied in this thesis and found suitable for the embedded microcontroller applications are presented in the remaining parts of this chapter.

5.4. The Caesar Cipher

One of the simplest and perhaps the smallest encryption algorithms is the Caesar Cipher as described below in figure 5.1.

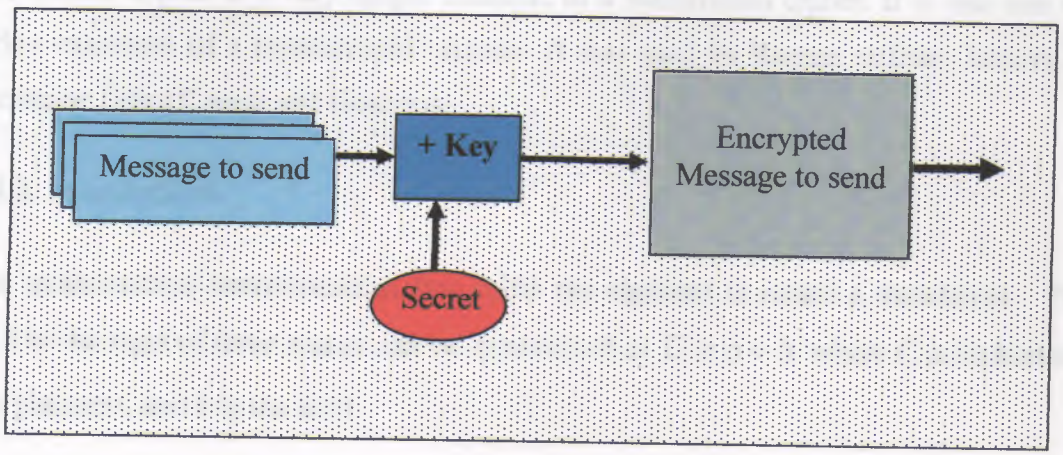


Figure 5.1 Caesar Cipher Message to send

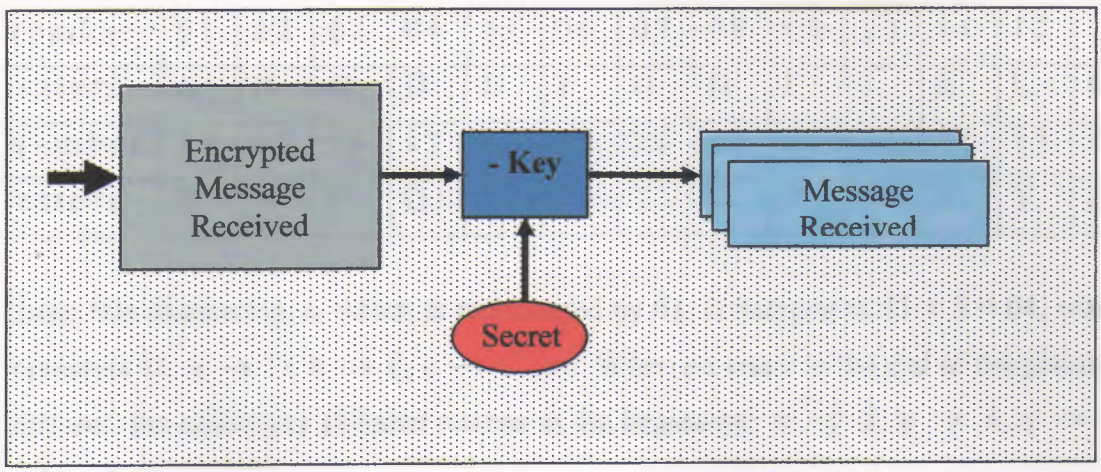


Figure 5.2 Caesar Cipher Encrypted Messages Received

5.4.1. The algorithm

The Caesar cipher encrypts a message by shifting every letter forward a specific number of places. For example, every letter could be shifted forward one place, so that 'a' is encrypted as 'B', 'b' as 'C', 'c' as 'D', and so on. As there are 26 letters in the alphabet, you can choose a shift of anything up to 25 places. (A shift of 26 takes each letter back to its original position, thus leaving each one unencrypted.)

To encrypt a message by applying a Caesar shift of one, for example, we would look up each letter in the top row and replace it with the corresponding letter in row 1. So the message 'defend east side', for example, becomes 'EFGFOE FBTU TJEF'.

The Caesar Cipher is a very simple example of a substitution cipher. It is also one the oldest examples of a cryptographic system. It was used by Roman military leaders, in particular Julius Caesar and Octavius.

5.4.2. Translations

The mathematical transformation that shifts the alphabet is called a translation. The shift to the right of three spaces can be symbolized as $C = p + b$ where C is ciphertext, p is plaintext, and b is the shift.

We think of assigning numbers to the letters of the alphabet. A usual numbering is

number	01	02	03	04	05	06	07	08	09	10	11	12	13
letters	A	B	C	D	E	F	G	H	I	J	K	L	M
Number	14	15	16	17	18	19	20	21	22	23	24	25	26
letters	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Using this numbering, a, which is represented by 01, is mapped to 01 + 3 = 04, which represents D. a → D, b → E, c → F, etc. When we come to the end of the plaintext alphabet, the ciphertext alphabet returns to the beginning: w → Z, x → A, y → B, and z → C.

Then the algorithm can be expressed as follows. For each plaintext letter p, substitute the ciphertext letter C:

Mathematically, we can express the Caesar Cipher as:

$C = E(p) = (p + 3) \bmod (26)$

A shift may be of any amount, so that the general Caesar algorithm is

$C = E(p) = (p + k) \bmod (26)$

Where k takes on a value in the range 1 to 25. The decryption algorithm is simply

$P = D(C) = (C - k) \bmod (26)$

The *key* for this Caesar cipher is 3 the number added to the plaintext letter to arrive at the corresponding ciphertext letter.

Here is a key that need not be written it can be remembered. Because we are adding the key to the numbers corresponding to the letters of the alphabet, we call this an additive key. In the case above, we say we are using an additive key of 3.

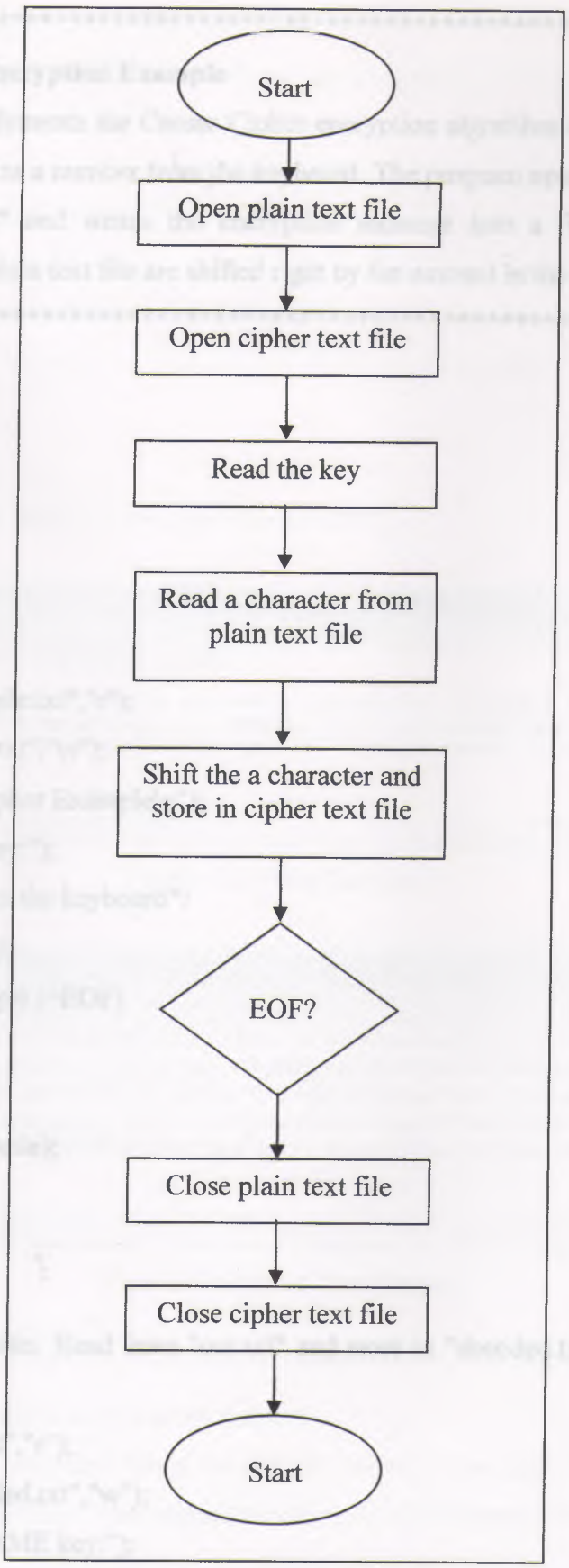


Figure 5.3 Caesar Cipher Flow Chart

```
/*********************************************************************
```

Caesar Cipher Encryption Example

This program implements the Caesar Cipher encryption algorithm on a PC. The key is entered as entered as a number from the keyboard. The program opens the plain-text file called "sample.txt" and writes the encryption message into a file called "out.txt". Characters in the plain text file are shifted right by the amount in the key.

```
*********************************************************************/

#include"stdio.h"
#include"stdlib.h"
main()
{
  char c,code,key;
  FILE *ifp,*ofp;
  ifp = fopen("sample.txt","r");
  ofp = fopen("out.txt","w");
  printf("Caesar Cipher Example\n");
  printf("\n Enter key:");
  /*Get the key from the keyboard*/
  scanf("%d",&key);
  while((c = getc(ifp)) !=EOF)
  {
  code = c + key;
  fprintf(ofp,"%c",code);
  }
  fclose(ifp);
  fclose(ofp);
/* Now see if it works. Read from "out.txt" and store in "decoded.txt" using the same
key value*/
  ifp= fopen("out.txt","r");
  ofp= fopen("decoded.txt","w");
  printf("\n Enter SAME key:");
  /*det the key*/
```

```
 scanf("%d",&key);
 while ((c=getc(ifp))!=EOF)
 {
 code=c-key;
 fprintf(ofp,"%c",code);
 }
 fclose(ifp);
 fclose(ofp);
 return 0;
 }
```

Figure 5.4 *Caesar Cipher PC Program (key used is 5)*

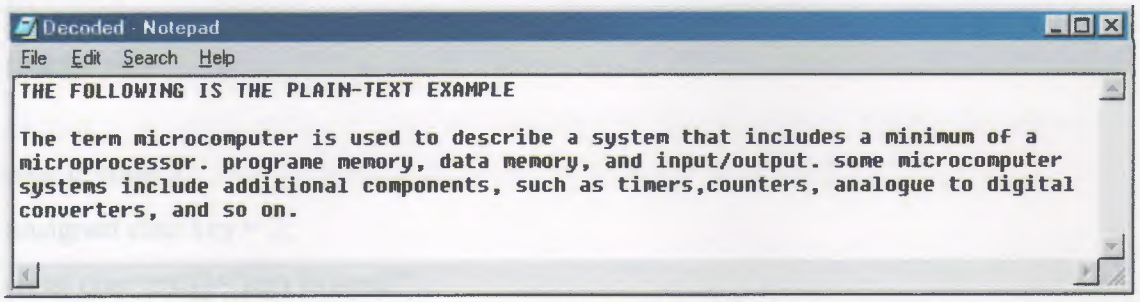


Figure 5.5 Plain Text Input

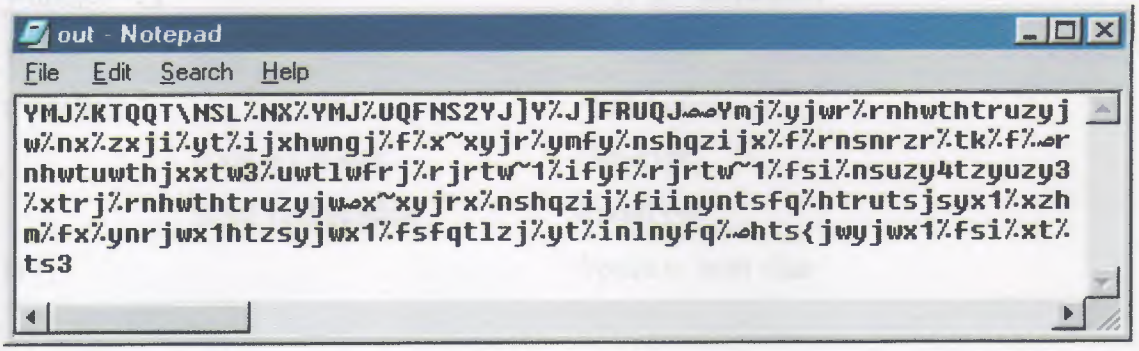


Figure 5.6 Cipher Text Output

```
/*****************************************************************
```

Microcontroller Program for Caesar Cipher

This program was compiled using the PIC C compiler the code is written for the PIC16F84 microcontroller. This microcontroller has 1k program memory and 64 byte RAM.

The text to be encrypted is stored in the program memory using the "const" declaration in order to save RAM space. Also, long text can be stored in the program memory. Fixed key value of 2 is used in this program.

This program occupies only 43 bytes in program memory, and 4 bytes in RAM memory. 13 bytes of the 43 are the text. Therefore, the size of the actual program (without the send_code routine is only 30 bytes).

**********/

```
#include<pic.h>
void send_code(char c)
{
}
main(void)
{
char c, code ,i;
unsigned char key = 2;
const char text[]="text to send!";
i=0;                                    //i=0
while(c!='!')                           //! Is the terminator
{
c=text[i];                              //get a char
            code = c + key;             //encrypt
            send_code(code);            //send out
i++;                                    //print to next char
   }
}
```

Figure 5.7 Caesar Cipher Microcontroller Program (Key=5)

A computer program was developed to simulate the Caesar Cipher on a PC. The flowchart of this program is given in Figure 5.3. As can be seen from the flow diagram, the implementation of the standard Caesar Cipher is extremely easy. The PC program is given in Figure 5.4. The program open a plain text file on the hard disk of the computer, reads characters from this file decodes the characters by shifting each character by the

amount specified in the key, and then writes the resultant Cipher text to an output file. The key is a number which specifies the amount of shift and is extend interactively from the keyboard. The plain text and the cipher text output from this program are shown in Figure 5.5 and 5.6 respectively. The simple Caesar Cipher was also implemented on a popular PIC16F84 model microcontroller, using the PIC C microcontroller compiler. The program simply encrypts some characters and sends them out from the microcontroller. The algorithm occupies only 43 bytes in program memory and only 4 bytes in RAM data memory. The microcontroller program listing is given in Figure 5.7.

5.5. Tiny Encryption Algorithm (TEA)

The Tiny Encryption Algorithm is one of the fastest and most efficient cryptographic algorithms in existence. It was developed by David Wheeler and Roger Needham at the Computer Laboratory of Cambridge University. [16] The author has dean use a short program which will run on most machines and encipher safely. It uses a large number of iterations rather than a complicated program as shown in figure 5.8.

It is hoped that it can easily be translated into most languages in a compatible way. The first program is given below. It uses little set up time and does a weak non linear iteration enough rounds to make it secure. There are no preset tables or long set up times. It assumes 32 bit words.

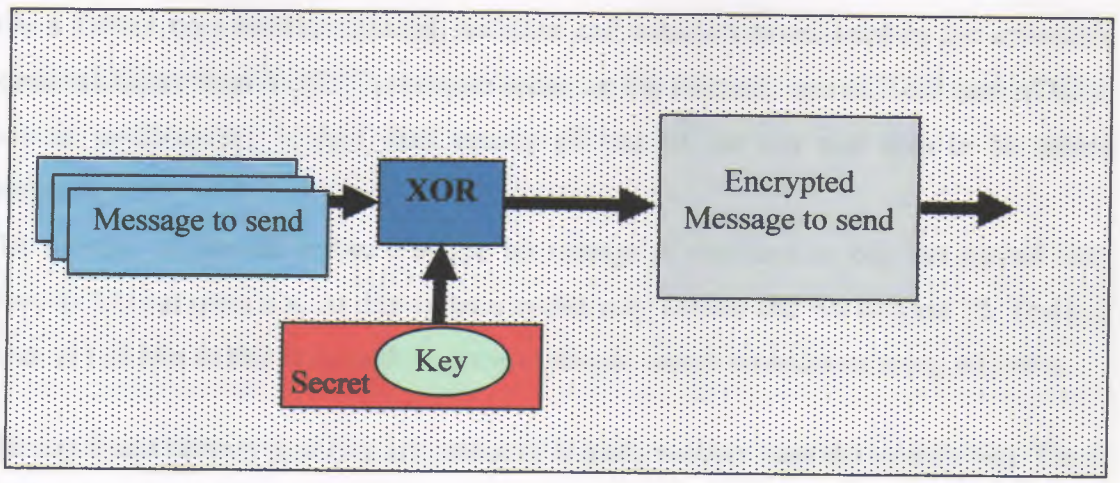


Figure 5.8 Tiny Encryption Algorithm (TEA) Encryption Message to Send

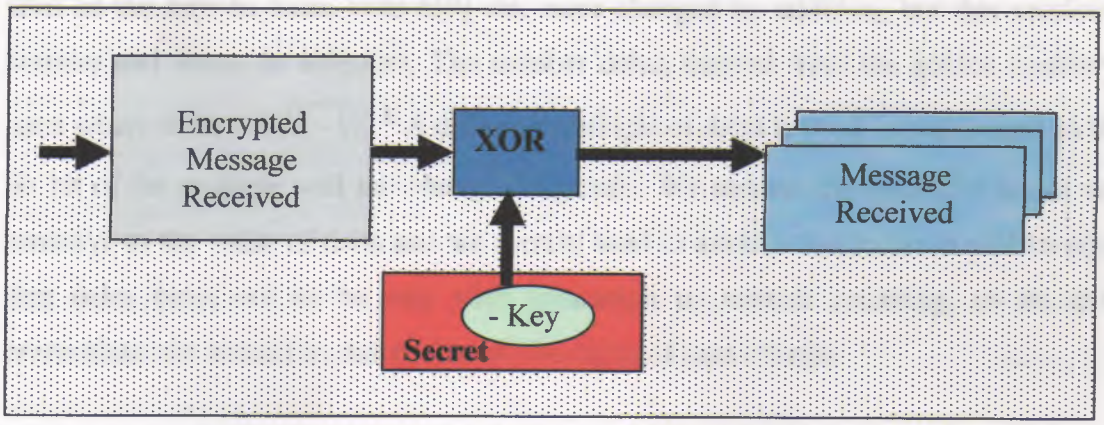


Figure 5.9 Tiny Encryption Algorithm (TEA) Encryption Message Received

5.5.1. The Algorithm

- Key 128 bits (k [0]..k [3]).
- Plaintext 64 bits (2 x 32 bits, text [0], and text [1]).
- In 32 rounds combines plaintext and key, swapping the two.
- Halves of plaintext.
- Uses reversible addition of unsigned integers, XOR (^).
- Combines plaintext with constant delta to obscure key.

5.5.2. Basics of the routine

It is a Feistel type routine although addition and subtraction are used as the reversible operators rather than XOR. The routine relies on the alternate use of XOR and ADD to provide nonlinearity. A dual shift causes all bits of the key and data to be mixed repeatedly.

The number of rounds before a single bit change of the data or key has spread very close to 32 is at most six, so that sixteen cycles may suffice and we suggest 32. The key is set at 128 bits as this is enough to prevent simple search techniques being effective.

The top 5 and bottom four bits are probably slightly weaker than the middle bits. These bits are generated from only two versions of z (or y) instead of three, plus the other y or z. Thus the convergence rate to even diffusion is slower. However the shifting evens this out with perhaps a delay of one or two extra cycles.

The key scheduling uses addition, and is applied to the unshifted z rather than the other uses of the key. In some tests k[0] etc. were changed by addition, but this version is simpler and seems as effective. The number delta, derived from the golden number is used where delta = $(\sqrt{5}-1)2^{31}$ A different multiple of delta is used in each round so that no bit of the multiple will not change frequently. We suspect the algorithm is not very sensitive to the value of delta and we merely need to avoid a bad value. It will be noted that delta turns out to be odd with truncation or nearest rounding, so no extra precautions are needed to ensure that all the digits of sum change.

The use of multiplication is an effective mixer, but needs shifts anyway. It was about twice as slow per cycle on our implementation and more complicated.

The use of a table look up in the cycle was investigated. There is the possibility of a delay are one entry of the table is used. For example if k [z&] is used instead of k[0], there is a chance one element may not be used of $(3/4)^{32}$, and a much higher chance that the use is delayed appreciably. The table also needed preparation from the key. Large tables were thought to be undesirable due to the set up time and complication. The algorithm will easily translate into assembly code as long as the exclusive or is an operation. The hardware implementation is not difficult, and is of the same order of complexity as DES, taking into account the double length key.

5.5.3. Tests

A few tests were run to detect when a single change had propagated to 32 changes within a small margin. Also some loop tests including a differential loop test to determine loop closures.

A considerable number of small algorithms were tried and the selected one is neither the fastest, nor the shortest but is thought to be the best compromise for safety, ease of implementation, lack of specialized tables, and reasonable performance. On languages which lack shifts and XOR it will be difficult to code. Standard C does makes an arithmetic right shift and overflows implementation dependent so that the right shift is logical and y and z are unsigned.

5.5.4. Usage

This type of algorithm can replace DES in software, and is short enough to write into almost any program on any computer. Although speed is not a strong objective with 32 cycles (64 rounds) on one implementation it is three times as fast as a good software implementation of DES which has 16 rounds.

The modes of use of DES are all applicable. The cycle count can readily be varied, or even made part of the key. It is expected that security can be enhanced by increasing the number of iterations.

5.6. Combining Caesar and Simple TEA

The original TEA algorithm is computer and requires 128 bits key with 64 bits of plain text. A modified and singles version of the TEA algorithm has been tired by the author XOR ring a key value with the characters to be encrypted. The flow diagram of this algorithm is given in figure 5.10. A program was developed on a PC to simulate the TEA algorithm. The program listing is given in Figure 5.11, where the key is accepted at the keyboard from the use. Figures 5.12 and 5.13, shows the plain text and the cipher text produced by this program. The simple TEA algorithm was also implemented on a PIC16F84 microcontroller and the program listing is given in Figure 5.14. This program occupies only 43 bytes in the program memory and only 4 bytes in the RAM data memory.

Another program was also divided by the author to simulate the behavior of the Caesar Cipher and the TEA algorithms. This is given in appendix A the plain text and the cipher text for these programs are also given in Appendix A.

The author has developed programs to simulate the behavior of combining the Caesar and simple TEA algorithm the program listing is given in Appendix B.

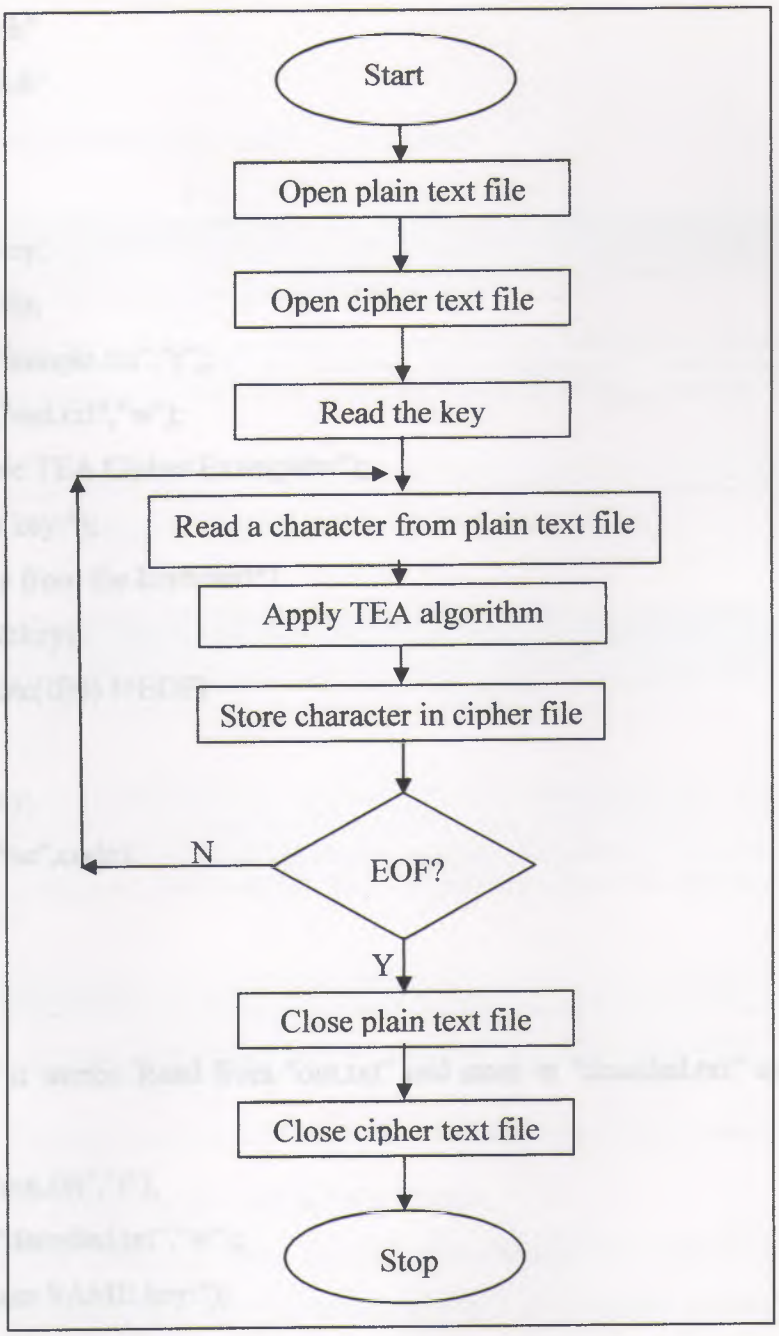


Figure 5.10 Simple TEA Flow Chart

/**

Simple TEA Cipher Encryption Example

This program implements the TEA Cipher encryption algorithm on a PC. The key is entered as entered as a number from the keyboard. The program opens the plain-text file called "sample.txt" and writes the encryption message into a file called "out.txt". Characters in the plain text file are shifted right by the amount specified in the key.

**/

```
#include"stdio.h"
#include"stdlib.h"
main()
{
  char c,code,key;
  FILE *ifp,*ofp;
  ifp = fopen("sample.txt","r");
  ofp = fopen("out.txt","w");
  printf("Simple TEA Cipher Example\n");
  printf("Enter key:");
  /*Get the key from the keyboard*/
  scanf("%d",&key);
  while((c = getc(ifp)) !=EOF)
  {
  code = c ^ key;
  fprintf(ofp,"%c",code);
  }
  fclose(ifp);
  fclose(ofp);
/* Now see if it works. Read from "out.txt" and store in "decoded.txt" using the same
key value*/
  ifp= fopen("out.txt","r");
  ofp= fopen("decoded.txt","w");
  printf("\n Enter SAME key:");
  /*det the key*/
scanf("%d",&key);
  while ((c=getc(ifp))!=EOF)
  {
  Code = c ^ key;
  fprintf(ofp,"%c",code);
  }
  fclose(ifp);
  fclose(ofp);
```

```
return 0;
}
```

Figure 5.11 Program listing of simple TEA ciphers

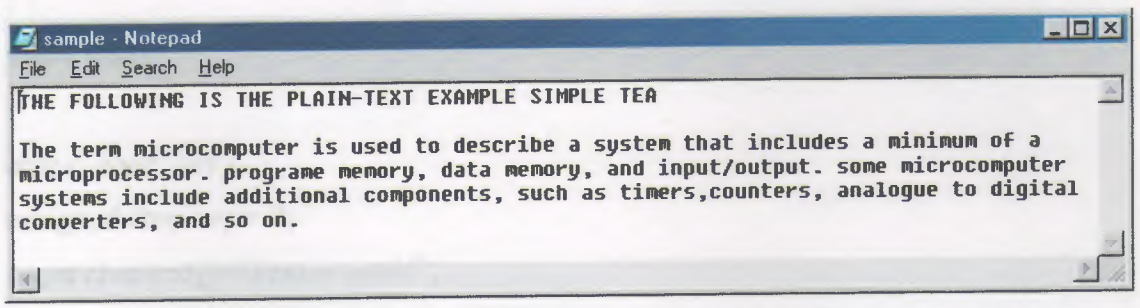


Figure 5.12 plain text for simple TEA cipher

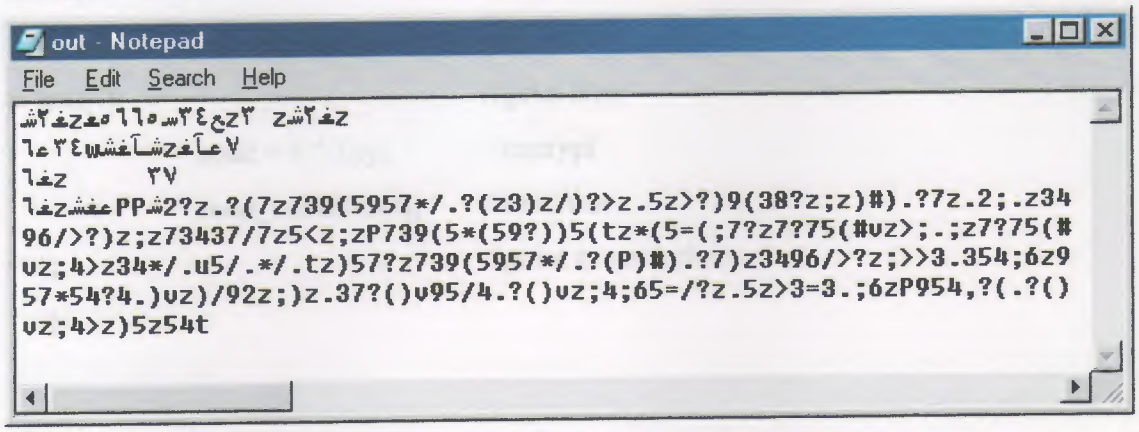


Figure 5.13 Cipher text for simple TEA cipher (key=90)

```
/*****************************************************************
```

Microcontroller Program for simple TEA Cipher

This program was compiled using the PIC C compiler the code is written for the PIC16F84 microcontroller. This microcontroller has 1k program memory and 64 byte RAM.

The text to be encrypted is stored in the program memory using the "const" declaration in order to save RAM space. Also, long text can be stored in the program memory.

Fixed key value of 2 is used in this program.

This program occupies only 43 bytes in program memory, and 4 bytes in RAM memory. 13 bytes of the 43 are the text. Therefore, the size of the actual program (without the send_code routine is only 30 bytes.)

```
*****************************************************************/
```

```
#include<pic.h>
void send_code(char c)
{
}
main(void)
{
char c, code ,key,i;
unsigned char key = 2;
count char text[]="Text to send!";
i=0;                                    //i=0
while(c != '!' )                        //! Is the terminator
{
c=text[i];                              //get a char
                code = c ^ key;         //encrypt
                send_code(code);        //send out
i++;                                    //point to next char
    }
}
```

Figure 5.14 Microcontroller implementation of simple TEA cipher

5.7. Using Variable key

The Caesar Cipher and the simple TEA algorithms are based a single key which can be crashed exactly. The author has developed a program which uses a variable key for each character in order to increase the security of the simple TEA algorithm. The program listing is given in Figure 5.17 have the key is incremented at each iteration; the microcontroller implementation of this algorithm is given in Figure 5.18.

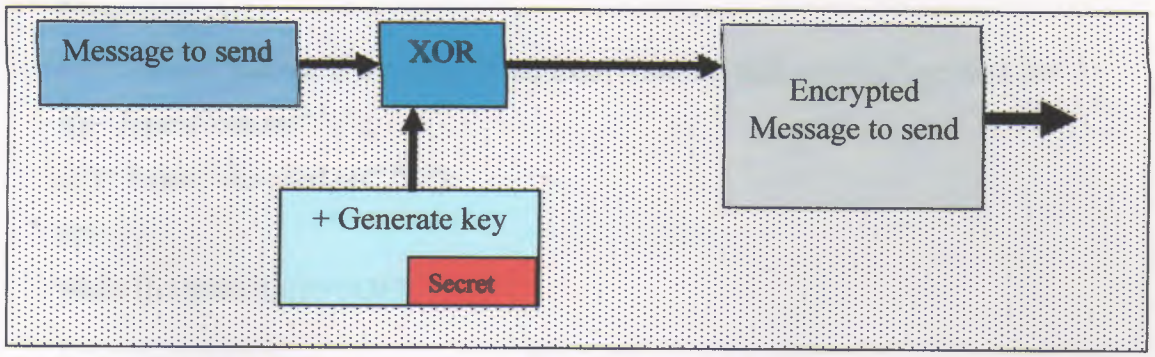


Figure 5.15 TEA Encryption Algorithm+ variable key for message to send

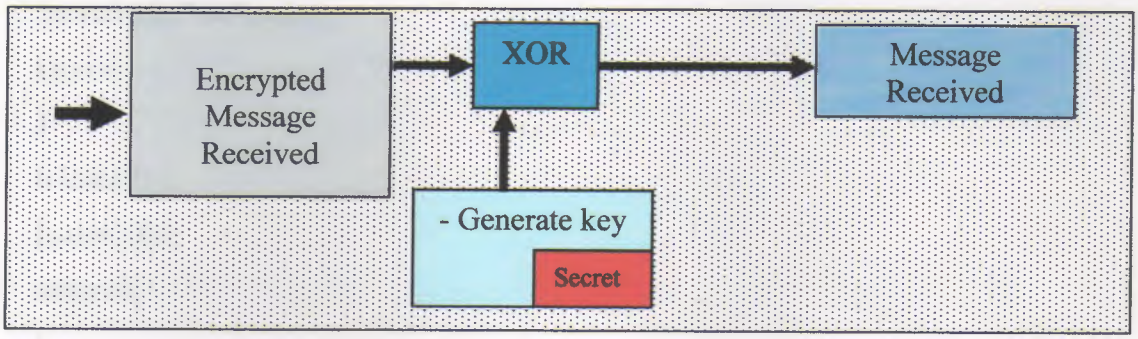


Figure 5.16 TEA Encryption Algorithm - variable key for Received message

```
#include"stdio.h"
#include"stdlib.h"
main()
{
/* First encode the data. Read from "Sample.txt and store in"out.txt"*/
  char c,code,key;
  FILE *ifp,*ofp;
  ifp = fopen("sample.txt","r");
  ofp = fopen("out.txt","w");
  key=1;
  while((c = getc(ifp)) !=EOF)
  {
  code = c ^ key;
  fprintf(ofp,"%c",code);
  key++;
  }
  fclose(ifp);
```

```
  fclose(ofp);
/*Now decode and see if it works. Read from "out.txt" and store in "decoded.txt" */
  ifp= fopen("out.txt","r");
  ofp= fopen("decoded.txt","w");
  key=1;
  while ((c=getc(ifp))!=EOF)
  {
  code=c^key;
  fprintf(ofp,"%c",code);
  key++;
  }
  fclose(ifp);
  fclose(ofp);
  return 0;
  }
```

Figure 5.17 simple TEA algorithms with varying key

```
/*********************************************************************
```

Encryption Using A 8 Bit Microcontroller.

This program was compiled using the PIC C compiler the code is written for the PIC16F84 microcontroller. This microcontroller has 1k program memory and 64 byte RAM. The text to be encrypted is stored in the program memory using the "const" declaration in order to save RAM space. Also, long text can be stored in the program memory.

This program occupies only 44 bytes in program memory, and 4 bytes in RAM memory. 13 bytes of the 44 are the text. Therefore, the size of the actual program (without the send_code routine is only 31 bytes).

```
*********************************************************************/
void send_code(char c)
{
} main(void)
{
```

```
char c, code ,key,i;
count char text[]="text to send!";
key = 1;                        //set the key
i=0;                            //i=0
while(c!='!')                   //! Is the terminator
{
c=text[i];                      //get a char
        code = c ^ key;         //encrypt
        send_code(code);        //send out
key++;                          //next key
i++;                            //point to next char
   }
}
```

Figure 5.18 Microcontroller Implementation of Simple TEA with Varying Key

5.8. Microcontroller implementation using a Timer to Create keys

The simple new algorithm "NEU" can be made key cure if a random key value is used for each character. The author has developed a program for microcontroller (PIC16F84) which uses the time of the microcontroller to get key values. The program listing is given in Figure 5.17 the time of the microcontroller is let to run continuously at rate of $2\,\mu s$. The keys are then read by the assignment key = TMRO. For each call to the time routine a different key is affianced and thus if becomes key difficult to break the code as shown in figure 5.17 & figure 5.18. If both the sender and the receiver use the same routines with the same microcontroller, then the same keys will be obtained.

The code given in Figure 5.21 can be improved by reading the time generated key values outside the encryption routine. This way, it will be easier to obtain the same key values for both the sender and the receiver systems. The new microcontroller program listing is given in figure 5.22.

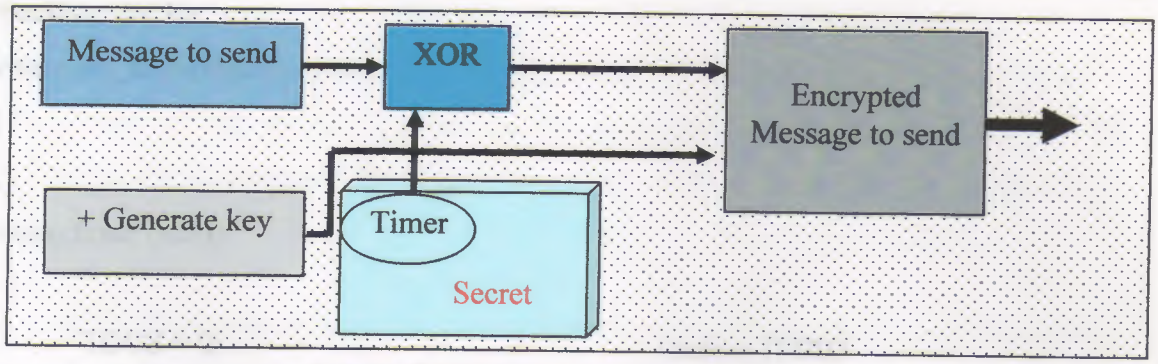


Figure 5.19 New Algorithms "NEU"

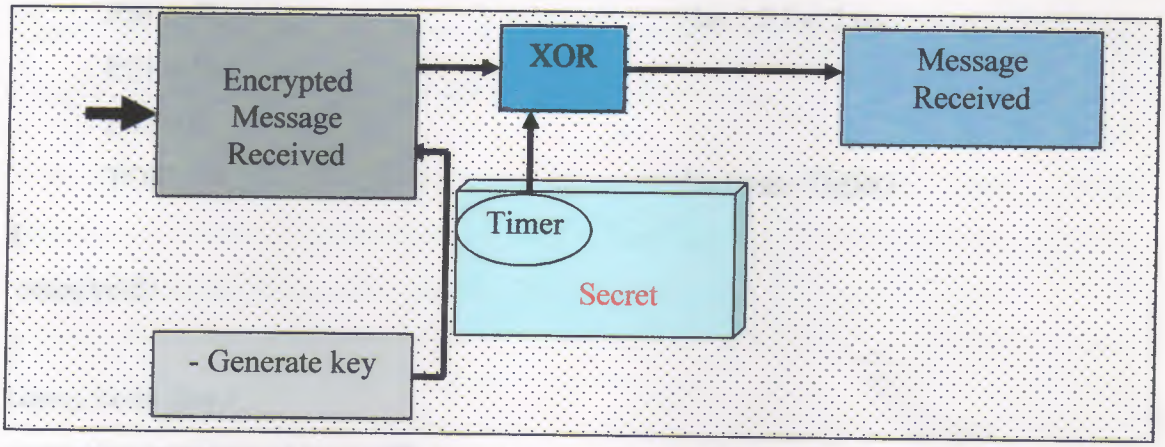


Figure 5.20 New Algorithms "NEU"

```
/*****************************************************************
```

Encryption Using A 8 Bit Microcontroller & Timer

This program was compiled using the PIC C compiler the code is written for the PIC16F84 microcontroller. This microcontroller has 1k program memory and 64 byte RAM.

The text to be encrypted is stored in the program memory using the "const" declaration in order to save RAM space. Also, long text can be stored in the program memory.

Timer TMRO is used to generate the keys.

This program occupies only 55 bytes in program memory, and 4 bytes in RAM memory. 13 bytes of the 44 are the text. Therefore, the size of the actual program (without the send_code routine is only 42 bytes.)

```
*****************************************************************/
```

```
#include<pic.h>
void send_code(char c)
{
}
setup_timer (void)
{
        TOCS = 0;                       //selecr TMRO
PSA = 0;                                //select TMRO pre-scaler
        PSO = 0;                        //as 1:2 i.e. 2 us
        PS1 = 0;
        PS2 = 0;
        TOIF = 0;                       //activate TMRO
}
 main(void)
{
char c, code ,key,i;
count char text[]="text to send!";
setup_timer();                          //select TMRO
TMRO=0;                                  //by loading a different
key = TMRO;                         //value to TMRO we can vary the first
                                        //key value
i=0;                                    //i=0
while (c != '!')                        //! Is the terminator
{
c=text[i];                              //get a char
            code = c ^ key;             //encrypt
            send_code(code);            //send out
key = TMRO;                             //get next key
i++;                                    //point to next char
  }
}
```

Figure 5.21 Microcontroller have limitations using a Timer

```
/*****************************************************************
```

Encryption of New algorithm "NEU" By Using A 8 Bit Microcontroller & Timer

This program was compiled using the PIC c compiler the code is written for the PIC16F84 microcontroller. This microcontroller has 1k program memory and 64 byte RAM.

The text to be encrypted is stored in the program memory using the "const" declaration in order to save RAM space. Also, long text can be stored in the program memory.

Timer TMRO is used to generate the keys. All the keys are read at the beginning of the program of the program to avoid any problems with timing.

This program occupies only 66 bytes in program memory, and 53 bytes in RAM memory. 13 bytes of the 66 are the text. Therefore, the size of the actual program (without the send_code routine is only 53 bytes.)

```
*****************************************************************/
#include<pic.h>
void send_code(char c)
{
}
setup_timer (void)
{
        TOCS = 0;                       //select TMRO
PSA = 0;                                //select TMRO pre-scalar
        PSO = 0;                        //as 1:2 i.e. 2 s
        PS1 = 0;
        PS2 = 0;
        TOIF = 0;                       //activate TMRO
}
 main(void)
{
char c, code ,i;
unsigned char keys [50];
count char text[]="Text to send!";
```

```
        setup_timer();                  //setups TMRO
TMRO=0;                                 //by loading a different value to TMRO we
                                        // can vary the first key value
for(i=0;i<50;i++)keys[i]=TMRO;          //get the keys
i=0;                                    //i=0
while (c != '!')                        //! Is the terminator
{
c=text[i];                              //get a char
            code = c ^ key[i];          //encrypt
            send_code(code);            //send out
i++;                                    //print to next char
  }
}
```

Figure 5.22 Getting the Time Generated Keys outside the Encryption Routine

5.9. Current Software and Hardware Cryptography

Most cryptographic algorithms are not designed with efficient software implementation as primary design criteria. For example, current workhorse encryption algorithms such as Triple-DES1 (3DES), requires 108 clock cycles per byte on a Pentium processor [39], yielding about ~9MBytes/second on a Gigahertz Pentium Pro. Likely successors to Triple-DES, the Advanced Encryption Standard (AES) candidates [40], all improve on the performance of Triple-DES, but still require 20-69 clock cycles per byte for 8 KB requests with an average penalty of an additional 3 cycles per byte a smaller, 1 KB request [39]. Hash functions have significantly better software performance than encryption. For example, SHA-1 on a 200MHz Pentium requires 13 clock cycles per byte (15 MB/second) while RIPE-MD160 hashes at 16 clock cycles per byte (12.5MB/second) [Preneel98]. While better than the fastest AES algorithms, they will still consume most of a 200 MHz Pentium's cycles supporting the media rates of current disk drives. These numbers show that a NASD-class processor, ~200 MIPS (e.g. 200 MHz Strong ARM), will be unable to support software-based cryptography, thus requiring hardware-based cryptography.

There are a wide range of hardware cryptographic accelerators. Eberlee at Digital's System Research Center demonstrated an experimental DES chip in 1992 that delivered 1 Gb/s performance [43]. Currently, you can purchase chips such as the Hi/Fn 7751 [42] or VLSI's VMS115 running at 80 MHz which deliver approximately 100 Mb/s and 200 Mb/s performance for both SHA-1 and Triple-DES. These chips, primarily designed to enable IPsec-based virtual private networks in 100Mb/second routers, may not be priced aggressively for commodity devices. Pijinburg Custom chips, next generation ASIC (500k gates, 0.18 micron) will implement SHA-1, Triple-DES, Safer SK64, and RIPEMD-160 [44] and is expected to deliver up to 500 Mb/s performance from each functional unit. Cognitive Designs next generation ASIC, the CDI 3000, will perform Triple-DES at 172 Mb/s and concurrent SHA-1 at 204 Mb/s, priced at approximately $20 in volume. While these cost and performance numbers are difficult to map directly into a NASD, they do provide an intuition of the performance and cost of readily available hardware support.

6. CONCLUSION

Microcontroller based security system and some cryptography algorithms have significance in this thesis. All the microcontrollers are working on some sort of encryption and decryption technique. This idea is implemented in many ways and by using many programming language softwares, like C/C++, JAVA, Visual Basic, Visual C++, etc.

I use a simple algorithm which can be translated into a number of different languages and assembly languages very easily. It is short enough to be programmed from memory or a copy. It uses a sequence of word operations rather than wasting the power of a computer by doing byte or 8 bit operations.

Caesar Cipher and the simple TEA can easily be implemented as an embedded microcontroller system, each occupying less than 50 bytes. The Caesar Cipher and TEA were combined to obtain an algorithm which is slightly more difficult to crash. The TEA algorithm was used with a varying key where different keys were used for each character. The resulting cipher text is more difficult to crash. The new version of the simple "NEU" algorithm is implemented. It runs on the microcontrollers and uses the timer routines to generate variable keys, making it very difficult to crash.

The table below summarizes the algorithms suitable for microcontrollers and their security levels.

Algorithm	**Security level**
Caesar Cipher Algorithm	low
TEA Encryption Algorithm	low
Caesar Cipher Algorithm + TEA Encryption Algorithm	Medium
TEA Encryption Algorithm+ variable key	Good
New Algorithm "NEU"	Very Good

REFERENCES

[1] Doğan Ibrahim (2000) "microcontroller Projects in C for the 8051" First published.

[2] Cryptography and Network Security, principles and practice Second Edition.

[3] W. Timothy Polk , John P. Wack, Lawrence E. BasshamIII, Lisa J. Carnahan "Anti-Virus Tools and Techniques for computer Systems".

[4] John Forrest Brown."Embedded Systems Programming in C and Assembly".

[5] Daniel D. Gajski, Frank Vahid, Sanjiv Narayan, Jie Gong ."Specification and design of Embedded Systems". University of California at Irvine.

[6] Hermann Kopetz.(1997). "Real-Time Systems Design Principles for Distributed Embedded Applications". Kluwer Academic Publishers, Boston/ Dordrecht/ London

[7] O.Goldreich. (1999). "Algorithms and Combinatorics. Modern Cryptography, Probabilistic Proofs and Pseudo-randomness"

[8] Josef Pieprzyk, Thomas Hardjono, Jennifer Seberry. "Fundamentals of Computer Security". [Springer].

[9] Charlie Kaufman, Radia Perlman, Mike Speciner. "Network Security Private Communication in a Public World".

[10] Sumit Ghosh. (2002). "Principles of Secure Network Systems Design".

[11] Kenneth L. Calvert, University of Kentucky, Michael J. Donahoo, Baylor University. (2001). "TCP/IP Sockets in Java Practical Guide for Programmers".

[12] Near East University. The Graduate Studies: A Complete reference. First Published 2001.

[13] Douglas E. Comer . "Internetworking With TCP/IP", Vol 1: principles, Protocols, and Architecture. Third Edition. Department of Computer Sciences, Purdue University, West Lofayette, In 47907.

[14] Laura A. Chapplell, Ed Tittel. "Guide to TCP/IP". Course Technology, Thomson Learning.

[15] Buck Graham. "TCP/IP Addressing, designing and Optimizing your IP Addressing Scheme".

[16] Bruce Schneier. (1996) "Applied Cryptography, Protocols, Algorithms and Sucre Code in C". 2d ed., John Wiley & Sons, New York.

[17] George Coulouris, Jean Dollimore, Tim Kindberg. "Distributed Systems, Concepts and Design".

[18] W.Stevens. "TCP/IP Illustrated ". www.google.com.

[19] Embedded TCP/IP Paper- Tracy Thomas at techonline.com

[20] Microchip manuals and Application Notes on Device Web-Connectivity

[21] S7600 – device specs, and program guide "www.google.com"

[22] Economic Factors "What is an Embedded System".www.eco.utexas.edu

[23] National Institute of Standards, Data Encryption Standard, Federal Information Processing Standards Publication 46. [January 1977].

[24] E. Biham and A. Shamir, Di_erential Springer-Verlag, (1993)"Analysis of the Data Encryption Standard".

[25] The Science of Secrecy http://www.channel4.com/science/microsites/S/secrecy

[26] Client Services Systems Administration. "htm:http://www.common-sense.com/servicesintro.html"

[27] Cryptology: *Albrecht Beutelspacher*. Mathematical Association of America, (1994).

[28] *Jon C. Graff.* Wiley, (2000). "Cryptography and E-commerce".

[29] Application Note 150 Small Message Encryption using SHA Devices. www.maxim-ic.com

[30] *Rodger* "Embedding PICmicro® Microcontrollers in the Internet". *Richey/Steve* Humberd,Microchip Technology Inc.Chandler, *AZ* http://www.sii.co.jp/compo.

[31] Caelli, W., Longley, D., and Shain, M., (1991). *Information Security Handbook,* Stockton Press, New York.

[32] Garfinkel, S., and Spafford, G., (1996). *"Practical UNIX and Internet Security"*. 2d ed., O'Reilly & Associates., Sebastopol, CA.

[33] Kaufman, C., Perlman, R., and Speciner, M., (1995). *"Network Security: Private Communication in a Public World"*. PTR Prentice-Hall, Englewood Cliffs, NJ,.

[34] National Research Council, *Computers at Risk: Safe Computing in the Information Age,* National Academy Press, Washington, D. C., (1991).

[35] Application Note 703 Embedded Networking with IPv6. www.maxim-ic.com

[36] Seiko Instruments Electronic Components."S-7600A Series iChip TCP/IP Protocol" http://www.seiko-usa-ecd.com/index.html

[37] Embedded-inet, Solutions.http://embedded-inet.com/

[38] Turley, Jim. "Embedded Processors by the Numbers." (1999). http://www.embedded.com

[39] Lehrbaum, Rick. "Linux and Windows Square Off over Devices." (2000). http://www.wideopen.com

[40] Schneier, and Whiting, D., Fast Software Encryption: Designing Encryption Algorithms for Optimal Software Speed on the Intel Pentium Processor, *Fast Software Encryption, Fourth International Workshop Proceedings*, [Springer – Verlag , 1997], pp. 242-259.24

[41] National Institute of Standards and Technology, Advance Encryption Standard Development Effort Webpage ttp://csrc.nist.gov/encryption/aes/aes_home.htm

[42] [Preneel98] Preneel, B., Rijmen, V., and Boosselaers, A., "Principles and performance of cryptographic algorithms," *Dr. Dobb's Journal*, Vol. 23, No., 12, [December 1998], pp. 126-131.

[43] Hi/fn, Inc. HiFn, 7751 Encryption Processor Data Sheet, [1999].

[44] Eberle, H., "A High-Speed DES Implementation for Network Applications," *Advances in Cryptology-- Proceedings of Crypto '92, Lecture Notes in Computer Science,* Springer Verlag, pp. 521-539.

[45] Van Pelt, P., Marketing Manager, Pijinenburg Custom Chips B.V., Personal Communication, [January 1999].

[46] Report finds 37 percent jump in security incidents Surge in mass-mailing worms fuels increase By James Niccolai. "InfoWorld" [April 04, 2003]

[47] Computer security incidents rise dramatically. The University of Queensland, Brisbane Australia http://www.uq.edu.au/ [06-Mar-2000]

[48] The Computer Emergency Response Team (www.cert.org) [1996-2001]

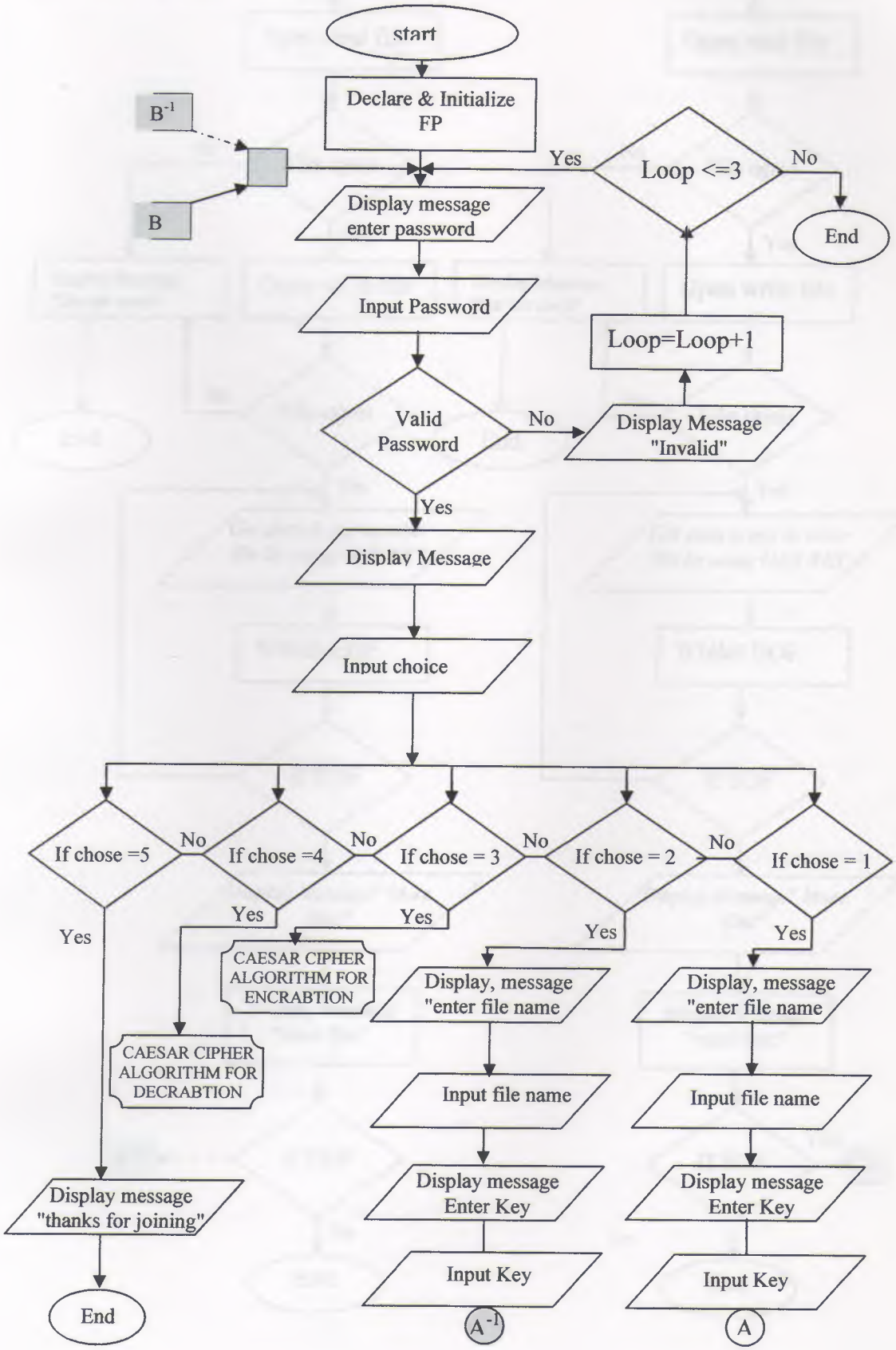
[49] Cyber Security, The Insurance Takeover http://www.ocipep-bpiepc.gc.ca/ critical /index_e.asp

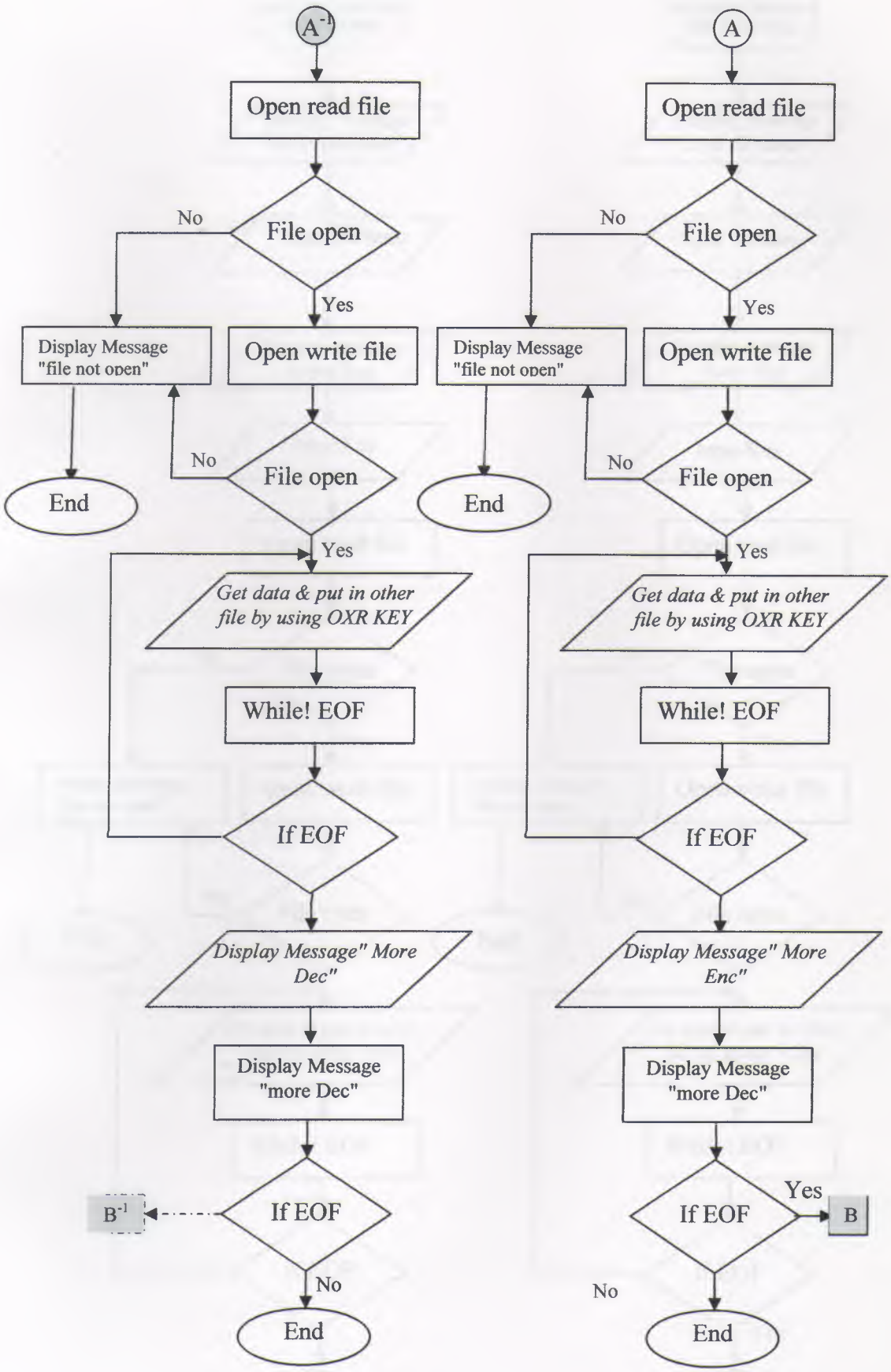
[50] Design feature Embedded-system security By Robert Monkman, OSE Systems. www.edn.com

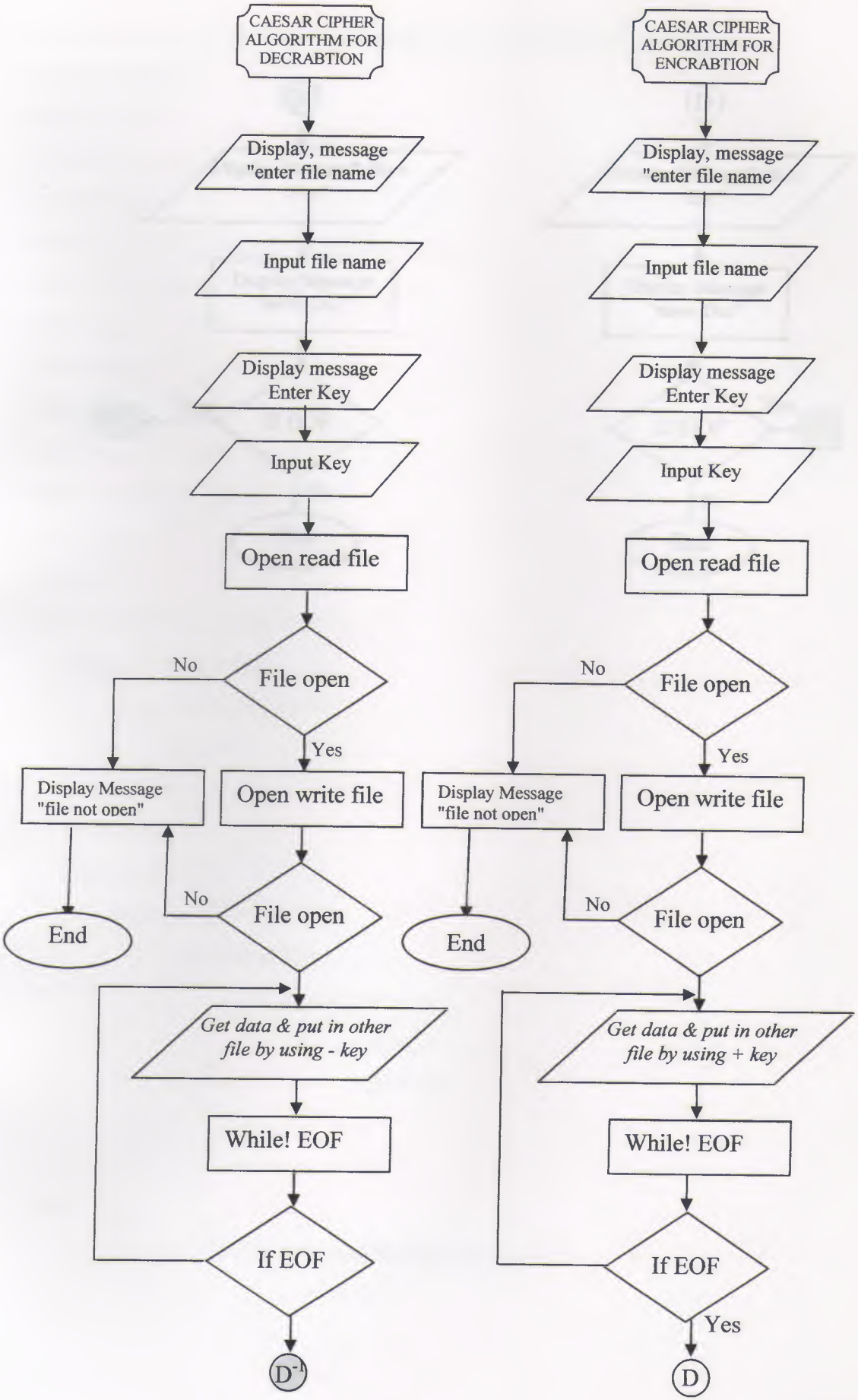
[51] SoC Solutions,www.xilinx.com/products/logicore/alliance/soc/soc.htm. President by James Bruister SoC Solutions, LLC jbruister@socsolutions.com

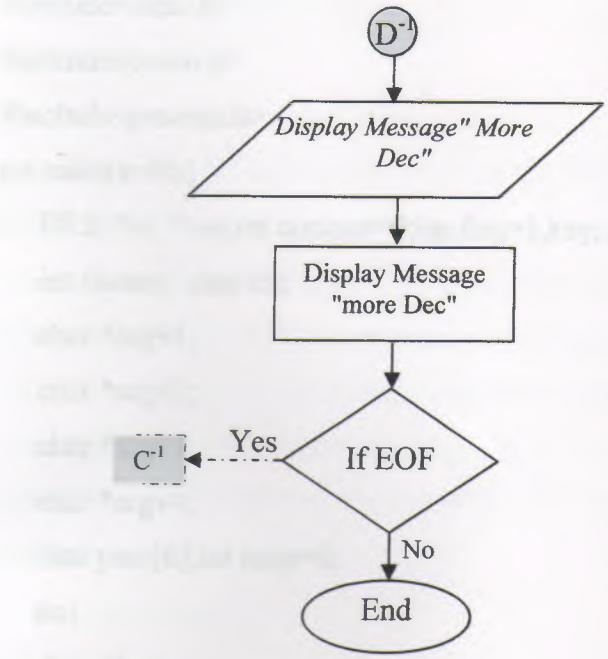
APPENDX I

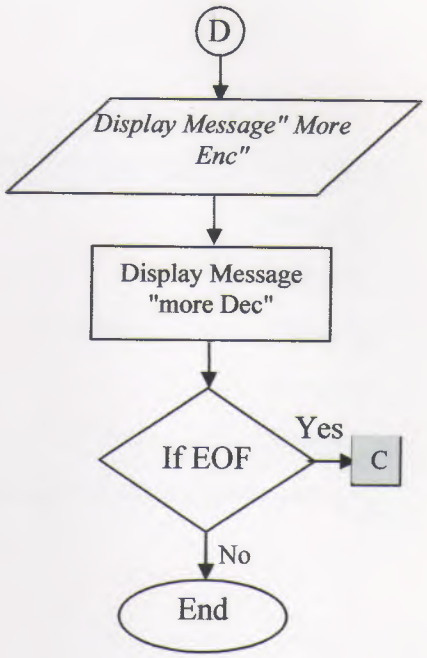
CAESAR CIPHER AND TEA FLOW CHART




A-1
Open read file
No
File open
Yes
Display Message "file not open"
Open write file
End
No
File open
Yes
Get data & put in other file by using OXR KEY
While! EOF
If EOF
Display Message" More Dec"
Display Message "more Dec"
B-1
If EOF
No
End
A
Open read file
No
File open
Yes
Display Message "file not open"
Open write file
End
No
File open
Yes
Get data & put in other file by using OXR KEY
While! EOF
If EOF
Display Message" More Enc"
Display Message "more Dec"
If EOF
Yes
B
No
End


CAESAR CIPHER ALGORITHM FOR DECRABTION
Display, message "enter file name
Input file name
Display message Enter Key
Input Key
Open read file
File open
No
Yes
Display Message "file not open"
Open write file
End
File open
No
Get data & put in other file by using - key
While! EOF
If EOF
D-1
CAESAR CIPHER ALGORITHM FOR ENCRABTION
Display, message "enter file name
Input file name
Display message Enter Key
Input Key
Open read file
File open
No
Yes
Display Message "file not open"
Open write file
End
File open
No
Get data & put in other file by using + key
While! EOF
If EOF
Yes
D


D-1
Display Message" More Dec"
Display Message "more Dec"
If EOF
Yes
C-1
No
End


D
Display Message" More Enc"
Display Message "more Dec"
If EOF
Yes
C
No
End

Program Encryption and Decryption of Caesar Cipher and Tea Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
int main(void){
  FILE *in, *out;int counter=0;int flag=1,key;
  int choice;  char ch;
  char *argv1;
  char *argv2;
  char *argv3;
  char *argv4;
  char pass[6];int loop=0;
  do{
  clrscr();
  printf("Enter password");
      while(counter<=4){
          pass[counter]=getch();
          counter++;
      }
      counter=0;
      if(pass[0]=='r'){
          if(pass[1]=='a')
              if(pass[2]=='e')
                  if(pass[3]=='d'){
                      getch();
                      goto Lable1;
          }
     }
  else{
                  printf("Invalid password");
                  getch();
                      }
loop++;
```

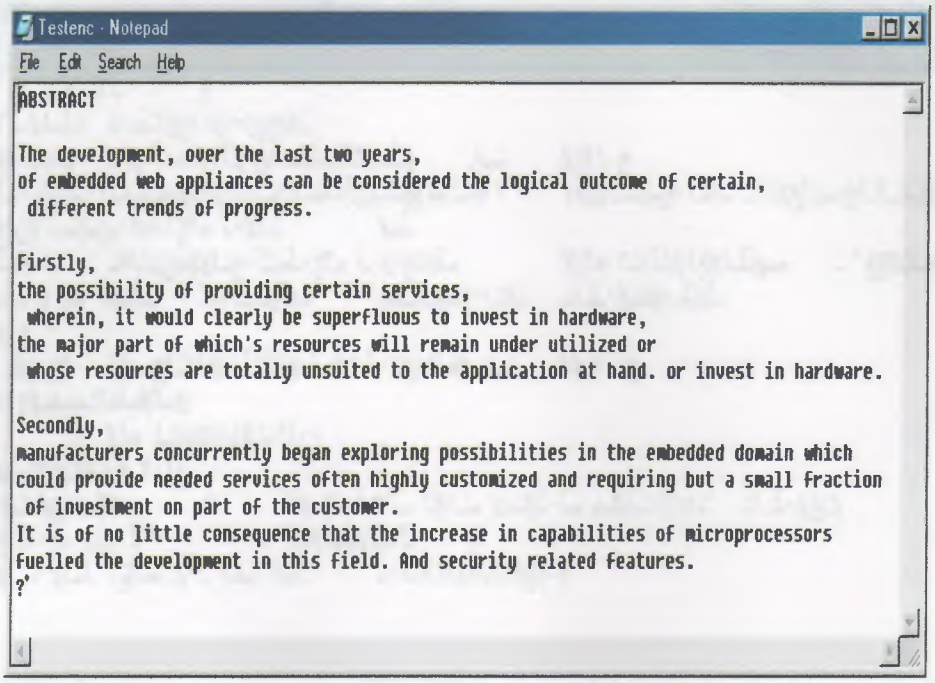
```
    }while(loop<3);
    if(loop==3)exit(0);
   Lable1:
    clrscr();
printf("\n\n **ENCRYPTION AND DECRYPTION FUNCTION MENU *****\n");
printf("\n  ***      1. Caesar Cipher Encryption data                    ***\n");
printf("\n  ***      2. Caesar Cipher Decrypt data                       ***\n");
printf("\n  ***      3. Encrypt data by TEA algorithm                    ***\n");
printf("\n  ***      4. Decrypt data by TEA algorithm                    ***\n");
printf("\n  ***      5. Exit from Program                                ***\n");
printf("\n  ************************************************************\n");
scanf("%d",&choice);
   switch(choice){
       case 1:
          printf("Enter Filename to encrypt: \t");
          scanf("%s",argv1);
          printf("%s",argv1);
          printf("\nEnter Encrypted fielname: \t");
          scanf("%s",argv2);
          printf("%s",argv2);
          printf("\nEnter Key:");
          scanf("%d",&key);
          if ((in = fopen (argv1,"r"))==NULL)
          {
             printf("Cannot open input file.\n");
             return 1;
          }
          else if ((out = fopen (argv2,"w"))== NULL)
             {
             printf("cannot open output file.\n");
             return 1;
             }
          else{
```

```
            while (!feof(in))
            fputc(fgetc(in)^key,out);
            }
            fclose(in);
            fclose(out);
            printf("\nData encrypted successfully");
            getch();
            printf("\nMore Enc");
            if(getch()==27 || getch()==27)
            exit(0);
            else
            goto Lable1;
            break;
case 2:
    printf("Enter Filename to Decrypt:\t");
    scanf("%s",argv1);
    getch();
    printf("%s",argv1);
    printf("\nEnter Decrypted fielname:\t");
    scanf("%s",argv2);
    printf("%s",argv2);
    printf("\nEnter Key:");
    scanf("%d",&key);
    getch();
    if ((in = fopen (argv1,"r"))==NULL)
    {
        printf("Cannot open input file.\n");
        return 1;
    }
     else if ((out = fopen(argv2,"w"))== NULL)
        {
        printf("cannot open output file.\n");
        return 1;
```

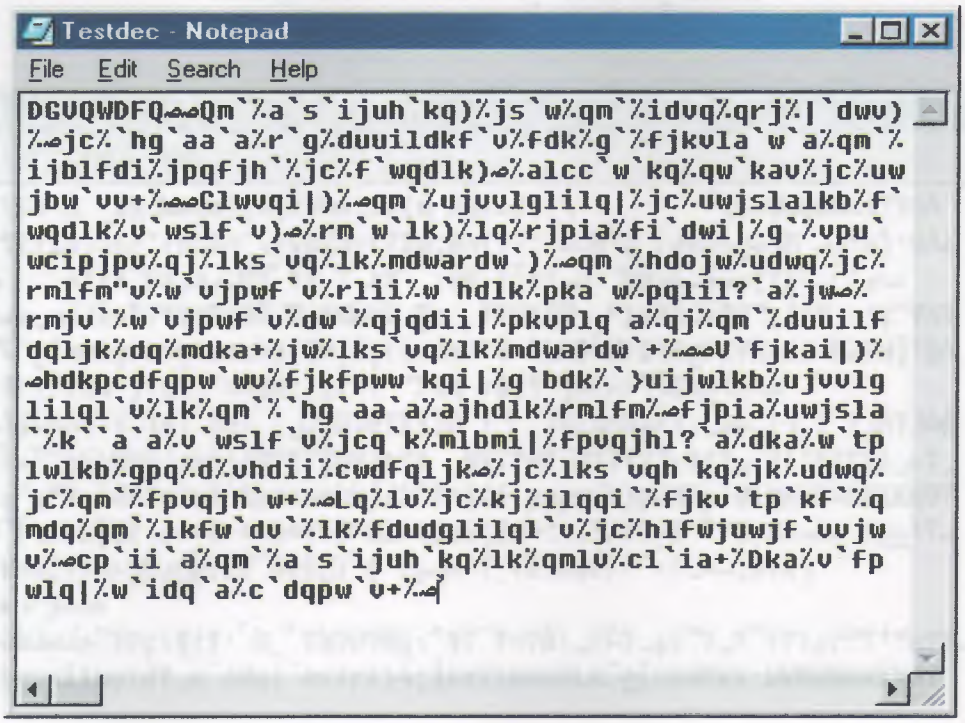
```
        }
        else{
        while (!feof(in))
        fputc(fgetc(in)^key,out);
        }
        fclose(in);
        fclose(out);
        printf("\nData Decrypted successfully");
        getch();
        printf("\nMore Enc");
        if(getch()==27 || getch()==27)
        exit(0);
        else
        goto Lable1;
    break;
case 3:
    printf("Enter Filename to encrypt: \t");
    scanf("%s",argv3);
    printf("%s",argv3);
    printf("\nEnter Encrypted fielname: \t");
    scanf("%s",argv4);
    printf("%s",argv4);
    printf("\nEnter Key:");
    scanf("%d",&key);
    if ((in = fopen (argv3,"r"))==NULL)
    {
        printf("Cannot open input file.\n");
        return 1;
    }
     else if ((out = fopen (argv4,"w"))== NULL)
        {
        printf("cannot open output file.\n");
        return 1;
```

```
        }
        else{
        while (!feof(in))
        fputc(fgetc(in)-key,out);
        }
        fclose(in);
        fclose(out);
        printf("\nData encrypted successfully");
        getch();
        printf("\nMore Enc");
        if(getch()==27 || getch()==27)
        exit(0);
        else
        goto Lable1;
        break;
case 4:
    printf("Enter Filename to Decrypt:\t");
    scanf("%s",argv3);
    getch();
    printf("%s",argv3);
    printf("\nEnter Decrypted fielname:\t");
    scanf("%s",argv4);
    printf("%s",argv4);
    printf("\nEnter Key:");
    scanf("%d",&key);
    getch();
    if ((in = fopen (argv3,"r"))==NULL)
    {
        printf("Cannot open input file.\n");
        return 1;
    }
        else if ((out = fopen(argv4,"w"))== NULL)
        {
```

```
            printf("cannot open output file.\n");
            return 1;
            }
            else{
            while (!feof(in))
            fputc(fgetc(in)+key,out);
            }
            fclose(in);
            fclose(out);
            printf("\nData Decrypted successfully");
            getch();
            printf("\nMore Enc");
            if(getch()==27 || getch()==27)
            exit(0);
            else
            goto Lable1;
            break;
    case 5:
       printf("\nHave a nice time");
       getch();
       system("exit");
       break;
    default:
            printf("\nInvalid choice");
            getch();
            break;
    }
    printf("\nThanks for joining us");
    getch();
    return 0;
}
```

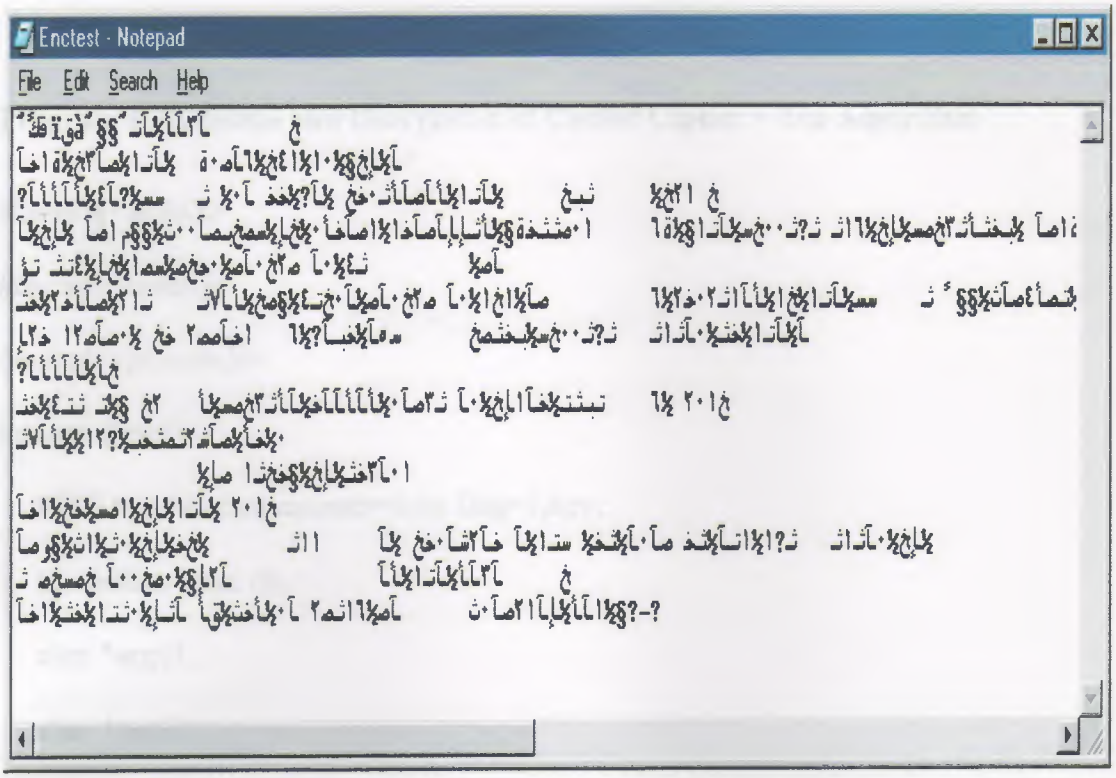


Example of Text File for Caesar Cipher Encryption Algorithm

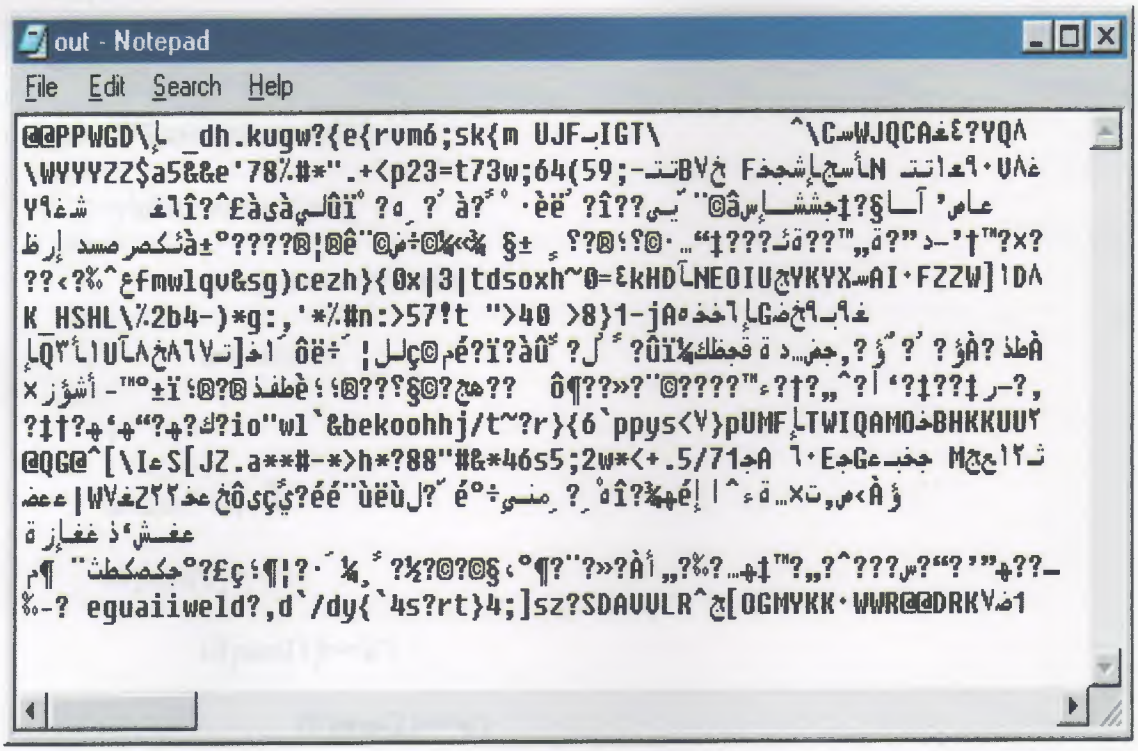


Encryption of Text File by Caesar Cipher Algorithm

Figure A.1 Caesar Cipher



Encryption of Text File by TEA Algorithm



Encryption Using EXOR with varying key

Figure A.2 TEA Cipher

APPENDIX II

Program Encryption and Decryption of Caesar Cipher + Tea Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
int main(void){
    FILE *in, *out;int counter=0;int flag=1,key;
    int choice;  char ch;
    char *argv1;
    char *argv2;
    char pass[6];int loop=0;
    do{
    clrscr();
    printf("Enter password");
        while(counter<=4){
                pass[counter]=getch();
                counter++;
        }
        counter=0;
        if(pass[0]=='r'){
                if(pass[1]=='a')
                        if(pass[2]=='e')
                                if(pass[3]=='d'){
                                        getch();
```

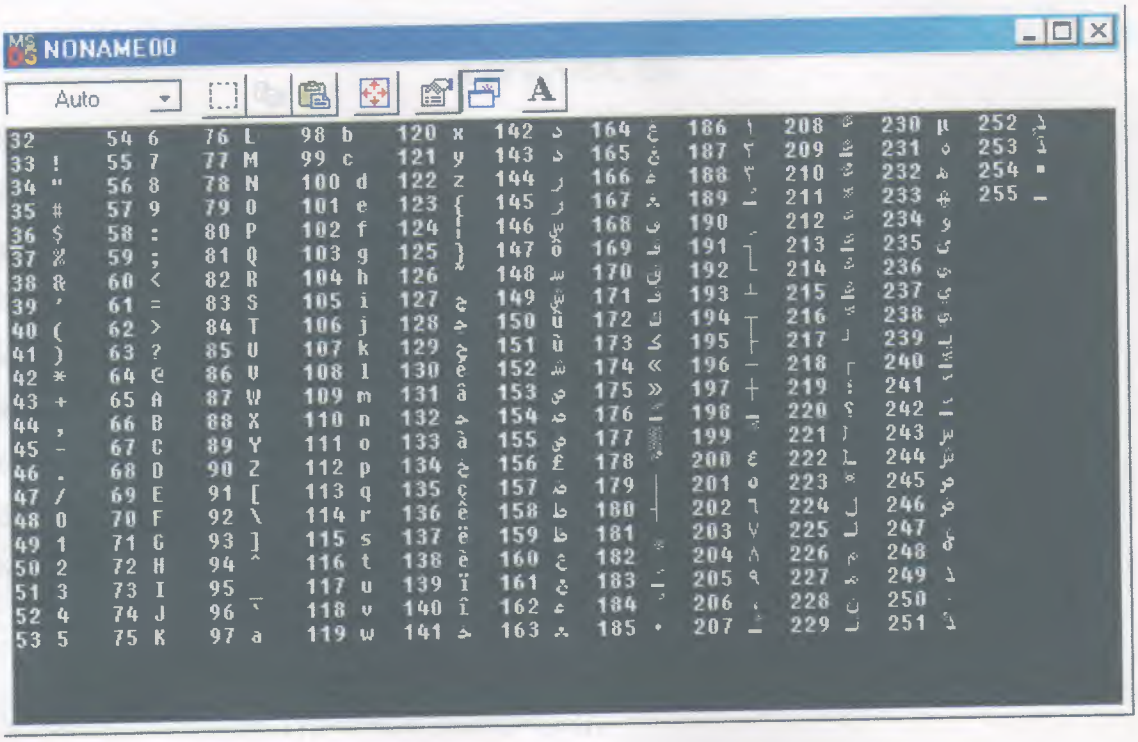
```
                    goto Lable1;
                }
            }
        else{
                    printf("Invalid password");
                getch();
                    }
        loop++;
        }while(loop<3);
       if(loop==3)exit(0);
      Lable1:
       clrscr();
printf("\n\n    ******* Encryption And Decryption FUNCTION Menu *********\n");
       printf("\n       ***          1. Encryption Algorthm                    ***\n");
       printf("\n       ***          2. Decryption Algorthm                    ***\n");
       printf("\n       ***          3. Exit from Program                      ***\n");
       printf("\n ***********************************************************\n");
       scanf("%d",&choice);
       switch(choice){
            case 1:
               printf("Enter Filename to encrypt: \t");
               scanf("%s",argv1);
               printf("%s",argv1);
               printf("\nEnter Encrypted fielname: \t");
               scanf("%s",argv2);
```

```
printf("%s",argv2);
printf("\nEnter Key:");
scanf("%d",&key);
if ((in = fopen (argv1,"r"))==NULL)
{
    printf("Cannot open input file.\n");
    return 1;
}
else if ((out = fopen (argv2,"w"))== NULL)
    {
    printf("cannot open output file.\n");
    return 1;
    }
 else{
    while (!feof(in))
    fputc((fgetc(in)+3)^key,out);
    }
    fclose(in);
    fclose(out);
    printf("\nData encrypted successfully");
    getch();
    printf("\nMore Enc'Yes','No'");
    if(getch()==27 || getch()==27)
    exit(0);
    else
```

```
            goto Lable1;
            break;
    case 2:
        printf("Enter Filename to Decrypt:\t");
        scanf("%s",argv1);
        getch();
        printf("%s",argv1);
        printf("\nEnter Decrypted fielname:\t");
        scanf("%s",argv2);
        printf("%s",argv2);
        printf("\nEnter Key:");
        scanf("%d",&key);
        getch();
        if ((in = fopen (argv1,"r"))==NULL)
        {
            printf("Cannot open input file.\n");
            return 1;
        }
          else if ((out = fopen(argv2,"w"))== NULL)
            {
            printf("cannot open output file.\n");
            return 1;
            }
            else{
            while (!feof(in))
```

```
            fputc((fgetc(in)^key)-3,out);
            }
            fclose(in);
            fclose(out);
            printf("\nData Decrypted successfully");
            getch();
            printf("\nMore Enc");
            if(getch()==27 || getch()==27)
            exit(0);
            else
            goto Lable1;
        break;
case 3:
        printf("\nHave a nice time");
        getch();
        system("exit");
        break;
    default:
            printf("\nInvalid choice");
            getch();
            break;
    }
    printf("\nThanks for joining us");
    getch();
    return 0;}
```

ASCII CODES of My Computer



32		54	6	76	L	98	b	120	x	142	د	164	غ	186	١	208	ﻫ	230	µ	252	ﻟ
33	!	55	7	77	M	99	c	121	y	143	ذ	165	ف	187	٣	209	ﻩ	231	٥	253	ﻠ
34	"	56	8	78	N	100	d	122	z	144	ر	166	ق	188	٣	210	ﻪ	232	٥	254	■
35	#	57	9	79	O	101	e	123	{	145	ز	167	ك	189	ـ	211	ﻫ	233	٥	255	
36	$	58	:	80	P	102	f	124	¦	146	ﺳ	168	ل	190	،	212	ﻭ	234	و		
37	%	59	;	81	Q	103	g	125	}	147	ﺷ	169	م	191	ﺁ	213	ﻯ	235	ى		
38	&	60	<	82	R	104	h	126	~	148	ﺻ	170	ن	192	ﺍ	214	ﻰ	236	ى		
39	'	61	=	83	S	105	i	127	⌂	149	ﺿ	171	ه	193	ﺏ	215	ﻱ	237	ى		
40	(	62	>	84	T	106	j	128	ﺀ	150	ﻁ	172	و	194	ﺕ	216	ﻲ	238	ى		
41	)	63	?	85	U	107	k	129	ﺁ	151	ﻅ	173	ى	195	ﺙ	217	ﻳ	239	ى		
42	*	64	@	86	V	108	l	130	ﺃ	152	ﻉ	174	«	196	ﺝ	218	ﻴ	240	ى		
43	+	65	A	87	W	109	m	131	ﺍ	153	ﻍ	175	»	197	ﺡ	219	ﻵ	241	ى		
44	,	66	B	88	X	110	n	132	ﺄ	154	ﻑ	176	▒	198	ﺥ	220	ﻶ	242	ى		
45	-	67	C	89	Y	111	o	133	ﺃ	155	ﻕ	177	▓	199	ﺩ	221	ﻷ	243	ى		
46	.	68	D	90	Z	112	p	134	ﺆ	156	£	178	│	200	ﺫ	222	ﻸ	244	ى		
47	/	69	E	91	[	113	q	135	ﻇ	157	ﻩ	179	┤	201	ﺭ	223	ﻹ	245	ى		
48	0	70	F	92	\	114	r	136	ﺈ	158	ﻁ	180	┤	202	ﺯ	224	ﻺ	246	ى		
49	1	71	G	93	]	115	s	137	ﺋ	159	ﻁ	181	╡	203	ﺱ	225	ﻻ	247	ى		
50	2	72	H	94	^	116	t	138	ﺋ	160	ﺟ	182	╢	204	ﺵ	226	ﻼ	248	ى		
51	3	73	I	95	_	117	u	139	ﻴ	161	ﺟ	183	╖	205	ﺹ	227	﻽	249	ى		
52	4	74	J	96	`	118	v	140	ﺊ	162	ﺀ	184	╕	206	ﺽ	228	﻾	250	ى		
53	5	75	K	97	a	119	w	141	ﺪ	163	ﺭ	185	╣	207	ﻁ	229		251	ى		

APPENDX III

```
#include "conio.h"
#include "stdio.h"
int K[10][10];int i,j;char Str[100]="NEAR EAST UNIVERISITY IS IN LEFKOSA.";int
loop=0;
int Enc[10][10];char Dec[10][10];
void main(){
//Kay[3]={2,2,2};
int Kay[10][10]={
                {27,05,17,17,05,17,17,05,04,14},
                {17,05,17,27,04,21,18,21,01,14},
                {45,05,37,17,06,02,02,19,02,13},
                {17,05,17,57,04,17,17,05,04,12},
                {14,05,47,47,05,21,18,21,07,11},
                {77,05,17,17,06,02,02,19,06,14},
                {19,05,57,37,05,17,17,05,07,17},
                {17,05,17,17,02,21,18,21,05,19},
                {16,05,37,27,01,17,17,05,07,14},
                {23,05,17,17,15,21,18,21,07,15}
                                        };
  char c,code;int count=0;
  FILE *ifp,*ofp;
  clrscr();
  ifp = fopen("sample.txt","r");
  ofp = fopen("out.txt","w");
  printf("Hill Caesar Cipher Example\n");
  while((c = getc(ifp)) !=EOF)
  {
  Str[count] = c ;
  fprintf(ofp,"%c",Str[count]);
```

```
    delay(10);
    count++;
    if(i==10 && j==10)
    {
    i=0;j=0;
    }
    if(count==100){
          delay(10);
          count=1;
    }
    }
    fclose(ifp);
//   fclose(ofp);
    printf("%d",count);
    delay(10);
clrscr();
for(i=0;i<10;i++){
for(j=0;j<10;j++){
printf("  %d    ",Kay[i][j]);
}
printf("\n");
getch();
}
printf("\n This is the Plaintext From File.\n ");
printf("\n");
for(i=0;i<10;i++)
for(j=0;j<10;j++){
K[i][j]=(int) Str[loop]*2;
loop++;
}
delay(10);
```

```
for(i=0;i<10;i++){
for(j=0;j<10;j++){
printf("  %d   ",K[i][j]/2);
}
printf("\n");
getch();
}
delay(10);
printf("\n This is the Encryption key\n ");
printf("\n");
//Multiplication.
for(i=0;i<10;i++){
for(j=0;j<10;j++){
Enc[i][j]=((K[i][j]*Kay[i][j])%26+45);
printf("  %d   ",Enc[i][j]);
}
printf("\n");
}
delay(10);
printf("\n This is the Encryption Output ");
printf("\n\n");
//Character.
for(i=0;i<10;i++){
for(j=0;j<10;j++){
printf("  %c   ",(char)Enc[i][j]);
}
printf("\n");
delay(10);
}
delay(10);
printf("\n This is the Output of Decryption ");
```

```
printf("\n\n");
//Digit.
for(i=0;i<10;i++){
for(j=0;j<10;j++){

Dec[i][j]=K[i][j]/2;
printf("%c",Dec[i][j]);
}
//printf("\n");
}
delay(10);
getch();
}
```