STUDY AND DESIGN OF SECURITY SYSTEM FOR AUDIOGRAPH MULTIMEDIA TEACHING SYSTEM

A thesis presented in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

at Massey University, Palmerston North,
New Zealand.

Shaohuai Zhang

2002

ABSTRACT

The AudioGraph is a project developed in the Institute of Information Sciences and Technology of Massey University as copyrighted software tools. The AudioGraph courseware files produced by the AudioGraph Recorder may be copyrighted. In order to protect them from been played back or copied by unauthorized parties, a security system to protect the AudioGraph courseware files is required.

This thesis presents the study, design and implementation of the AudioGraph security system. The security system proposed in this thesis consists of three parts: *Copy Protection Record* inside the AudioGraph courseware files; a Key Insertion Tool to detect, extract, insert and update the *Copy Protection Record* in AudioGraph courseware files; and a scheme of usage control embedded into the AudioGraph Plug-in.

The issues covered in this thesis include all relevant aspects. In order to select good encryption algorithms for the AudioGraph security system, this thesis introduces the concept of cryptography and describes some of the most important conventional and public-key encryption algorithms. It also investigates and compares various aspects of some of the conventional cryptography algorithms and chooses very strong, simple and suitable encryption algorithms to be used in the AudioGraph security system. A scheme to protect AudioGraph courseware files is described in this thesis, this scheme meets the requirements of the AudioGraph security system, and it is strong enough to withstand brute-force attack and all known cryptanalysis.

The implementation of the AudioGraph security system is also been described in this thesis. The result from system testing demonstrates that this AudioGraph security system works well and had achieved its goal to protect the AudioGraph courseware material.

ACKNOWLEDGEMENT

This Master Thesis is for my new life in New Zealand. There has been so much encouragement and support over the past few years, I am really grateful for it.

A special thanks goes to my supervisor, Professor Chris Jesshope. Thank you for your support and encouragement. I have never gained so much self-confidence before.

A hearty thank you goes out to my wife and my parents for their continued support. Without them, I could not have finished this thesis.

Thanks goes to the staff in computer science department of Massey University, without their encouragement and support, I could not have finished the research work over the past few years.

I also want to thank all my colleagues in the New Zealand Educational Software unit. It makes me feel happy working in such a good team.

Shaohuai Zhang
Master of Science (Computer Science) Candidate,
Massey University

Computer Science,
Institute of Information Science and Technology
Massey University
Palmerston North
New Zealand

LIST OF ACRONYMS

AES	Advanced Encryption Standard
API	Application Programming Interface
CA	Certification Authority
DES	Data Encryption Standard
DLL	Dynamic-Link Library
GSM	Global System for Mobil telecommunication
HTML	HyperText Makup Language
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
IDEA	International Data Encryption Algorithm
IETF	Internet Engineering Task Force
ITU	International Telecommunications Union
ITU-T	ITU Telecommunications Standardization Sector
KDC	Key Distribution Center
LFSR	Linear-Feedback Shift Register
MIME	Multipurpose Internet Mail Extensions
NIST	National Institute of Standards and Technology of America
NSA	National Security Agency (the official security body of the U.S. government)
PNG	The Portable Network Graphics format
SQA	Software Quality Assurance
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TEA	Tiny Encryption Algorithm
TLS	Transport Layer Security

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

This chapter briefly describes the concept of multimedia teaching and the AudioGraph project in the Institute of Information Sciences and Technology of Massey University, and introduces some of the technologies used in the AudioGraph project. This project has looked at the problem of introducing protection system into the playback of the Audiograph multimedia files. The concepts of computer security and cryptography, which are related to the protection system of AudioGraph, are introduced and the requirements for the protection system of the AudioGraph are both described in this chapter.

1.1 Introduction of Multimedia Teaching

Teaching is an interactive goal-directed activity, which includes presentations given by the lecturer. In a presentation, the lecturer usually explains abstract concepts or theories and demonstrates ideas by using all available tools – oral description, drawing a diagram, showing a working model etc. Usually the students will ask questions about the presentation, do exercises or experiments and the lecturer will answer questions, correct the mistakes of the students on their exercises, instruct on the experiments etc.

It is obvious that all of these teaching activities would need to include interactive audio, visual, text communication and animation. With the invention of the computer and the widespread use of multimedia and computer networks, especially the Internet, all of these activities connected with teaching can be realized locally or distantly. Therefore the student and the lecturer do not need to meet together at a specific place and time in order to be instructed on a specific topic. Self-paced, individual-time-and-space-independent learning becomes possible. Moreover, multimedia computer presentations make it possible to visualize abstract concepts and simulate a process, thus making it easier to understand. With artificial intelligence integrated into the multimedia education system, a student may carry out experiments and interact with virtual worlds and an intelligent agent may act as if it were a lecturer. There are a growing number of electronic interactive textbooks that are now available on the net. More recently there

has been a large interest, in the university sector, in the establishment of virtual universities (HART & MANSON, 1996).

The success of Multimedia Teaching systems depends critically on available hardware technologies: high performance CPU and I/O bus architectures, high-quality audio & visual input/output devices, large capacity data-storage devices and high-speed networks. In addition, these systems require a much higher level of system software support in comparison to past computational and communicational environments. The recent personal computer system has advanced so fast that it now makes multimedia teaching possible. However, there are still problems with network speed, because most potential 'virtual' students do not sit on an intranet but are at the end of a, possibly quite slow, modem line. Multimedia teaching courseware and authoring tools therefore must take account of this problem.

The AudioGraph project addresses this problem by providing tools to develop interactive multimedia presentations, which stream over low bandwidth connections (Jesshope et al., 1998).

1.2 Overview of AudioGraph Multimedia Teaching Tools

AudioGraph is a project that was initially developed in the Computer Science Department of Massey University. The AudioGraph system includes multimedia tools that have been developed for recording audio-graphic presentation material for publication in an HTML reference environment. At present, the tools comprise Macintosh applications (Audiograph Recorder) and Netscape plug-ins. Figure 1.1 shows the AudioGraph tools and their relationship to each other.

The AudioGraph Recorder will be used to record the lectures including text, audio and graphic information and produce AudioGraph courseware files (".aep" files and corresponding HTML documents). The Plug-ins support the playback of presentations in the AudioGraph file format, in Web browsers that support Netscape's Plug-in interface.

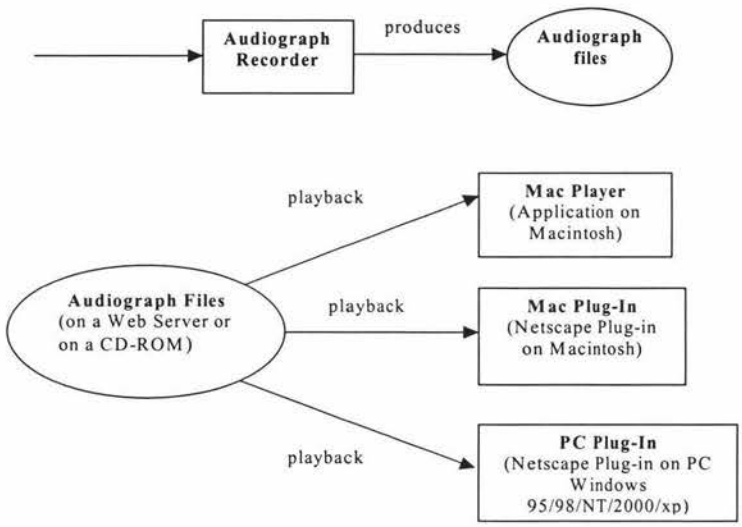


Figure 1.1: Tools AudioGraph

In order to playback AudioGraph courseware, the AudioGraph Plug-In should be put into the "Plugins" directory of Netscape browser or corresponding plug-in directory of other Web Browser that supports the Netscape Plug-in interface. The AudioGraph courseware files can be made available in one of two formats:

- on CD-ROM
- on the Web Server

As show in the following diagram:

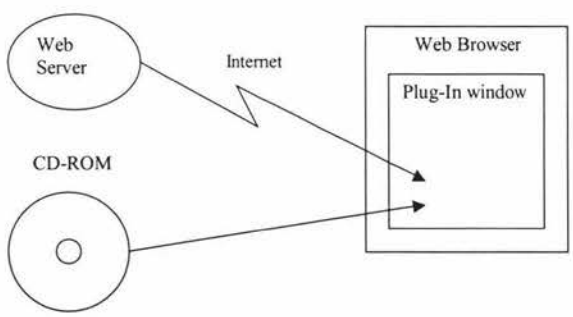


Figure 1.2: Courseware Delivery

This project is concerned with introducing Protection Control into the playback of AudioGraph presentations, so that only authorized or registered users may be able to access the material. Because the playback can be from CD, as well as from the Web, then the normal web-server security techniques are not sufficient to provide the usage

protections that we require. Before the specific issues of Protection Control are discussed, this section looks at techniques and standards used in the AudioGraph Plug-in. In particular the Netscape plug-in interface and two compression standards used in delivering AudioGraph files, namely PNG image compression and GSM sound compression. More details of the AudioGraph project are available in (New Zealand Educational Software, 2000).

1.3 Overview of the Techniques used in the AudioGraph

The PC Plug-in is a tool to playback the recorded AudioGraph courseware on the Web browser. The PC Plug-in extends the supported MIME types of the Web browser in this case to support the playback of the AudioGraph file type. In January of 1996, Netscape Communications Corp. introduced the Plug-in API, which allowed the Web browser to be extended to support user defined MIME types. The Netscape Plug-In API is discussed in this section.

As has been mentioned before, one of the problems with distance, multimedia education is the need to transmit large amounts of courseware over the network, and the available network speed for the 'virtual' student is relatively slow. In order to let the 'virtual' student sit at the end of a slow modem line and still to be satisfied with the delivery speed of the courseware, it is necessary to compress the courseware as much as possible to maintain the user's satisfaction from the quality of sound, text and picture information. The techniques used to compress sound and picture are also reviewed in this section.

1.3.1 Netscape Plug-In API

Netscape's Plug-In API was first introduced by Netscape Communicator corp. along with the beta release of its Navigator 2.0 Web browser (Oliphant, 2000). Since then, many major companies have accepted this standard and have created plug-ins to support their file formats. Now, both the Netscape Navigator and Microsoft Internet Explorer support Netscape's Plug-In API. (Note: for some reasons, Microsoft no longer supports Netscape's Plug-In API after its Internet Explorer 5.5 SP2).

The Web browser is related to the Multipurpose Internet Mail Extensions (MIME). In the early days of ARPANET, email comprised only text messages. Nowadays email on the Internet includes audio, video message and support for other languages like French, Russian and Chinese as well. The MIME standard was created for supporting data types including images, audio, video, binary data, fonts, and text in the electronic mail. MIME is now widely used not only in email but also over the Internet to identify any file that can be transmitted over a network. Today's Web browser supports many MIME types. HyperText Makup Language (HTML) is one of the most popular MIME type. The MIME type is usually associated with one or more file extensions. For instance, a HTML files will have an extension of .HTML or .HTM. When a file is sent over a network, the server uses a MIME table to associate a MIME type with the file extension, the MIME type is then sent with the data stream over the network. Our AudioGraph files will have a file extension ".aep" and MIME type "application/vnd.audiograph". A Netscape Plug-in will extend the supported MIME types of a Web browser to include this type. When the corresponding Plug-in is registered with the browser, the browser extends its capability to the new MIME type. The complete description of MIME is in RFC1341 (Borenstein & Freed, 1992) and RFC1521 (Borenstein & Freed, 1993).

In Windows 95/98/NT, a Plug-in is implemented as a Dynamic-Link-Library (DLL) that has a name starting with NP. The MIME type information for a Plug-in is defined by the file version resource information. In the file version resource, the resource key of MIMEType defines the MIME type, and the resource key of FileExtents defines the associated file extension of the MIME type. In the version resource, some other information can be defined, for instance, the ProductName will define the name of the Plug-In, and the FileDescription will give a description of the Plug-In.

A Windows Plug-In DLL has only three official library entry points: NP_GetEntryPoints, NP_Initialize and NP_Shutdown. Inside the NP_GetEntryPoints definition, more plug-in entry points are defined. The NP_Initialize will initialize the DLL, and NP_Shutdown is called before the DLL is shutdown. Netscape has designed entry points of the Plug-in to simplify the cross-platform code. The Plug-in implemented entry points, or methods as Netscape calls them, are prefaced with NPP_. Table 1.1 is the currently available Plug-in implemented methods.

Table 1.1: Plug-In implemented method (Netscape Communications Corporation, 1997a)

Method Name	Description
NPP_Destroy	Deletes a specific instance of a plug-in.
NPP_DestroyStream	Tells the plug-in that a stream is about to be closed or destroyed.
NPP_GetJavaClass	Returns the Java class associated with the plug-in.
NPP_GetValue	Allows Communicator to query the plug-in for information.
NPP_HandleEvent	Delivers a platform-specific window event to the instance.
NPP_Initialize	Provides global initialization for a plug-in.
NPP_New	Creates a new instance of a plug-in.
NPP_NewStream	Notifies a plug-in instance of a new data stream.
NPP_Print	Requests a platform-specific print operation for an embedded or full-screen plug-in.
NPP_SetValue	Sets information about the plug-in.
NPP_SetWindow	Tells the plug-in when a window is created, moved, sized, or destroyed.
NPP_Shutdown	Provides global deinitialization for a plug-in.
NPP_StreamAsFile	Provides a local file name for the data from a stream.
NPP_URLNotify	Notifies the instance of the completion of a URL request.
NPP_Write	Delivers data to a plug-in instance.
NPP_WriteReady	Determines maximum number of bytes that the plug-in can consume.

The Navigator-implemented Plug-in methods are prefaced with NPN_. Table 1.2 is the currently available Navigator-implemented Plug-in methods.

A detailed description of the Netscape Plug-in API is available from Web Site of Netscape Communicator Corp. (Netscape Communications Corporation, 1997b).

Table 1.2: Navigator-implemented Plug-In method (Netscape Communications Corporation, 1997a)

Method Name	Description
NPN_DestroyStream	Closes and deletes a stream.
NPN_ForceRedraw	Forces a paint message for a windowless plug-in.
NPN_GetJavaEnv	Returns a pointer to the Java execution environment.
NPN_GetJavaPeer	Returns the Java object associated with the plug-in.
NPN_GetURL	Requests Communicator to create a stream for the specified URL.
NPN_GetURLNotify	Requests creation of a new stream with the contents of the specified URL; gets notification of the result.
NPN_GetValue	Allows the plug-in to query Communicator for information.
NPN_InvalidateRect	Invalidates specified drawing area prior to repainting or refreshing a windowless plug-in.
NPN_InvalidateRegion	Invalidates specified drawing region prior to repainting or refreshing a windowless plug-in.
NPN_MemAlloc	Allocates memory from Communicator's memory space.
NPN_MemFlush	Requests that Communicator free a specified amount of memory.
NPN_MemFree	Deallocates a block of allocated memory.
NPN_NewStream	Requests the creation of a new data stream produced by the plug-in and consumed by Communicator.
NPN_PostURL	Posts data to a URL.
NPN_PostURLNotify	Posts data to a URL, and receives notification of the result.
NPN_ReloadPlugins	Reloads all plug-ins in the `Plugins` directory.
NPN_RequestRead	Requests a range of bytes for a seekable stream.
NPN_SetValue	Sets windowless plug-in as transparent or opaque.
NPN_Status	Displays a message on the status line of the browser window.
NPN_UserAgent	Returns Communicator's `user agent` field.
NPN_Version	Returns version information for the Plug-in API.
NPN_Write	Pushes data into a stream produced by the plug-in and consumed by Communicator.

After successfully building a Netscape Plug-in DLL, to install it, just copy the DLL files into Netscape's Plug-in subfolder – the *Plugins* subfolder. When Netscape Navigator is launched, it enumerates all the files whose name start with NP in the *Plugins* subfolder and queries each Plug-in for its MIME type, description and suffixes. Once Netscape Navigator is running, the user can see what Plug-ins are available by selecting the *Help | About Plug-ins* menu, then all active Plug-ins including their MIME type are displayed on the Netscape Navigator. Figure 1.3 show MIME type information of PC Plug-in of AudioGraph from Netscape Navigator:

Figure 1.3: MIME type information of AudioGraph PC Plug-in from Netscape Web Browser

The MIME type must be added to the Web server so that the Web server can recognize this MIME type and files of this type based on the file extension. When a Web browser requests a file, the Server responds to this by sending the MIME type of the file and then streaming file's data to the browser.

The PC Plug-in of AudioGraph is an embedded plug-in. When a Web browser makes a request to the Web Server, the AudioGraph HTML document must contain a TAG like:

```
<EMBED SRC="bvn1.aep" WIDTH="663" HEIGHT="492" ALIGN="BOTTOM">
```

to request the display of AudioGraph courseware file "bvn1.aep" using the Plug-in. This causes the browser to request file "bvn1.aep" from the Web server and the server firsts looks-up the MIME type based on the file extension. Then, if it finds the file extension ".aep" has been entered in the MIME type table, it starts sending file "bvn1.aep" with the MIME type to the browser. The browser then checks whether it can handle this type. It will find that the file with extension ".aep" will be handle by a Plug-in DLL "NPaep.dll", then it will load the DLL to handle this file. The result is to playback the

AudioGraph course material in file "bvn1.aep" within the region specified inside the <embed> tag.

1.3.2 PNG image compression

The Portable Network Graphics (PNG) format was designed to replace the older and simpler GIF format and, to some extent, the much more complex TIFF format. It is used in two major fields: on displaying images over the Web as well as in editing and storing images. It provides a portable, legally unencumbered, well-compressed, well-specified standard for lossless, bitmapped image files.

The PNG supports three image types: true-colour (up to 48 bits per pixel), grayscale (up to 16 bits per pixel) and 8 bits palette-based images. GIF only supports palette-based image types, and JPEG only supports true-colour and grayscale image types. Compared with GIF, PNG has four advantages: full alpha channels (i.e. variable transparency, GIF has only binary transparency), image gamma correction (supports automatic cross-platform control of image brightness/contrast), a faster initial presentation using a two-dimensional interlacing (a method of progressive display), and straightforward detection of file corruption. PNG also compresses around 5% to 25% better than GIF in almost every case. However, GIF supports multi-images, but PNG only support a single image format. The following section overviews some of the technical details of PNG, the full specification of PNG is available on (Boutell & Lane, 1996).

File Structure

A PNG file consists of a *File Signature* followed by a series of *Chunks.*

The File Signature contains the following 8 byte fixed value (given here as a decimal encoding): 137 80 78 71 13 10 26 10. Its purpose is to detect file corruption due to file transmission over the Network especially across different systems, where different standards, such as for end of line markers, may have been automatically applied to the data.

The structure of each *Chunk* in a PNG file is the same, it consists of four parts:

1. *Length*: 4-bytes unsigned integer, giving the number of bytes in the *chunk data* field of a chunk.
2. *Chunk type*: 4-bytes, giving the type of a chunk.
3. *Chunk data*: the actual data appropriate to the chunk type, variable length. The byte number of this field is giving by the *Length* field of this chunk.
4. *CRC*: the cyclic redundancy check calculated on *chunk type* code and *chunk data* field.

The chunk series will start with IHDR chunk and end with IEND chunk. There are two different chunk categories: Critical Chunk and Ancillary Chunk. Some of the critical chunks are as following:

- IHDR – Image header
- PLTE – Palette chunk
- IDAT – Image data
- IEND – Image trailer

Some of the Ancillary chunks are as following:

- bKGD – Background colour
- cHRM – Primary chromaticities and white point
- gAMA – Image gamma
- hIST – Palette histogram
- iCCP – Embedded ICC profile
- pHYs – Physical pixel dimemsions
- sBIT – Significant bits
- sPLT – Suggested palette
- sRGB – Standard RGB colour space
- tEXt – Textual data
- tIME – Image last-modification time
- tRNS – Transparency
- zTXt – Compressed textual data

Compression

PNG compression method 0 is the only compression method currently supported. It is a Lempel-Ziv 77 (LZ77) (Emeterio, 1999) derivative used in the popular zip, gzip, pkzip. It uses the zlib compression engine which is defined in RFC1950 (Deutsch, 1996).

To make the image compression better, compression filters may be used to transform the image before it is compressed. Each horizontal line can be associated with one of the five possible filters. The filter itself doesn't reduce the file size of the image, but it can improve the compressibility. The article in (Roelofs, 1996a) gives an example – an extreme case – to reduce image file size by using a compression filter:

"a 512 x 32,768 image containing all 16,777,216 possible 24-bit colours compressed **over 300 times better** with filtering than without. The uncompressed image was 48 MB in size; the compressed-but-unfiltered version was around 36 MB; but the filtered version is only 115,989 bytes (0.1 MB)."

In PNG, the compression method and compression filter is specified in the IHDR chunk.

Alpha Channels

An Alpha channel is used to represent the transparency of a pixel. All of the three image types can use alpha values, but it is most often used with true-colour or grayscale images. For true-colour images, a pixel with an alpha channel is represented as RGBA (Red, Green, Blue, Alpha). An alpha value of zero means full transparency, i.e., the image is fully transparent (invisible) and all of the background is fully visible. An alpha value of $2^{bitdepth-1}$ represents a fully opaque pixel. Therefore the user can use the Alpha channel to combine the displayed image with the background image and to yield a composite image. Compared with GIF's support of only fully transparency or fully opaque pixels, PNG's variable transparency is a great advantage to the user.

Gamma Correction

Images on different computer monitors may have different setting for brightness/contrast, for instance, a Macintosh generated image tends to look too dark on PC, even a image generated on a PC may not look right on other PC.

In PNG, the gAMA chunk is used to specify a power function for relating the desired display output. In the gAMA chunk, a gamma value is specified and the display output intensity and sample value has the following relation:

```
Sample = light_out ^ gamma
```

However, the gamma value is only an approximation of the display device. The better approximation is to use the so-called chromaticity values, which are supported in PNG by a cHRM chunk. The best solution is to use the Colour Management System, which is supported in PNG by a sRGB chunk.

Interlacing

PNG supports interlacing. The user can decide to store an image file in interlaced order to allow progressive display. The interlacing method 0 is actually no interlacing, it just stores the pixels in a scanline left to right, and the scanlines from top to bottom. The interlace method 1 is known as Adam7. Adam7 consists of 7 distinct passes over the image, each pass transmits a subset of the pixels in an image according to the following 8×8 pattern over the whole image:

```
1 6 4 6 2 6 4 6
7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6
7 7 7 7 7 7 7 7
3 6 4 6 3 6 4 6
7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6
7 7 7 7 7 7 7 7
```

The number in the above pattern means the pass number to transmit this pixel. By using this interlacing method, the features of the image are presented faster. Usually any embedded text is readable after pass two and that only requires transmitting 2/64 of the whole image.

File Integrity Checks

PNG uses three methods for file integrity check: *File Signature*, *CRC-32* and *Adler-32 Checksum*.

The *File Signature* is a fixed 8 bytes containing the following value:

```
Decimal: 137   80   78   71   13   10   26   10
ASCII  : \211  P    N    G    \r   \n   \032 \n
```

The purpose of this *File Signature* is to quickly check if the file is corrupted due to transmission in Text mode. For example, over a Unix system, the "\r" byte in the *File Signature* will be lost and over a Macintosh system, the \n byte will be lost. In this way, the decoder will be able to detect the corruption of a file by just reading these 8 bytes.

The *CRC-32* checksum is a 4-bytes field (32 bits) in every chunk of a PNG file. The *CRC-32* algorithm is defined by ISO3309 and is popular used in network protocol for the purpose of error detection. In the PNG format, the CRC-32 field is calculated over the chunk type and chunk data inside a chunk. With CRC-32, the decoder of the PNG file will be able to detect any error in the chunk type or chunk data.

The *Adler-32 checksum* is calculated over the uncompressed stream data to make sure the integrity of the whole raw image data stream. It is a lower level checksum.

This section introduces some aspect of PNG format, the interested reader can find more details about PNG images in (Roelofs, 1996b).

1.3.3 GSM sound compression

The GSM sound compression algorithm is the sound compression algorithm adopted in Global System for Mobil telecommunication (GSM). Its original purpose is to compress speech signals for digital cellular phones. Now GSM has been used in many multimedia software applications to compress sound messages (Degener, 2000).

The GSM sound compression algorithm is a lossy, telephone-quality sound compression algorithm. The algorithm will compress a buffer of 160 bytes of raw sound data block into a GSM frame which is 32.5 bytes long. In practice, the GSM encoding algorithm usually takes 320 bytes of raw sound data block and compresses it into a double GSM frame which is 65 bytes long.

More detail on GSM speech compression algorithm is available from article on (Degener & Bormann, 2000a). The source code of the GSM algorithm in C language is unlicensed and freely available in (Degener & Bormann, 2000b). The source code in Java language is also freely available in (Degener & Bormann, 2000c) but, unlike the C library, the Java code is licensed under the Free Software Foundation's General Public License.

1.4 Introduction to Computer Security

Before the widespread use of computer systems and computer networks, sensitive information within an organization was held in the format of paper files stored in filing cabinets, as well as in the minds of the staff. Therefore information security was provided by physical means such as a lock to protect the paper files, and the administrative means to hire and manage reliable people.

With the widespread use of computer systems, the requirement to protect sensitive data, software and other valuable assets within the computer systems of an organization has become evident. The generic name for the collection of tools designed to protect data and to thwart hackers is *Computer Security*. The following introduces some concepts concerned with computer security.

1.4.1 Security Service

There is no standard definition of security services. In the opinion of the author, the definitions in (Stallings, 1995b) are accurate and complete. According to (Stallings, 1995b), the security service can be classified as:

Confidentiality: Requires that the information in a computer system and transmitted information be accessible for reading by only the authorized parties.

Authentication: Requires that the origin of a message be correctly identified, with an assurance that the identity is not false.

Integrity: Requires that computer-system assets and transmitted information can only be modified by authorized parties.

Nonrepudiation: Requires that neither the sender nor the receiver of a message be able to deny the transmission.

Access Control: Requires that access to information resources be controlled by or for the target system.

Availability: Requires that computer system assets be available to authorized parties when needed.

1.4.2 Security Attack

A security attack is the action that compromises the security of information in the computer or on the network. Security attacks can be classified into two major categories: active attack and passive attack.

An active attack includes *interruption*, *modification* and *fabrication*. *Interruption* is an attack on *availability*, example includes cutting off a communication line between the source and destination so that the information is not available by the receiver. *Modification* and *fabrication* are attacks on *integrity*. Examples include modifying the contents of the message on the network, insertion of spurious messages in a network.

Passive attack includes *Interception* and *traffic analysis*. *Interception* is an attack on *confidentiality*, e.g., the unauthorized party gains access to an asset for example a password file stored in the system diskette.

The party that makes the security attack may be a person or a piece of software (include attack tool, worm, virus etc.).

1.4.3 Cryptography in Computer Security

Cryptography plays a major role in computer security. Cryptographic algorithms serve nearly all the security services. Using a cryptographic algorithm to encrypt the sensitive information means that only the parties who have the correct key can decrypt the information and read it, therefore *Confidentiality* is achieved. In the UNIX system, *Access Control* is achieved by a password system which uses a variant of DES encryption algorithm. Security Services are achieved by using *cryptographic protocol*. *Cryptographic protocol* is the protocol that uses cryptography algorithm to archive a security task. A *cryptographic protocol* is a series of steps, involving two or more parties, designed to accomplish a security task. There are many different cryptographic protocols, designed to achieve different security services. For *Cryptographic protocol*, the interesting readers can refer to (Schneier, 1996).

In cryptography, the original message is called *plaintext* (or *cleartext*). The process of disguising a message so that its substance is hidden is called *encryption*. After being encrypted, the message becomes *ciphertext*. The process of turning ciphertext back into plaintext is called *decryption*. In the past, the security of an algorithm was based on keeping the algorithm secret, i.e., keeping the mechanism of the algorithm secret. The problems with secret algorithms are obvious, for instance, when someone accidentally reveals the secret of the algorithm, everyone using that algorithm must change it. It would take much time to choose a good algorithm and to changeover to that algorithm. Another problem with secret algorithms is that every product that does not share the secret should use their own secret algorithm, as a result, there would be too many different algorithms, and that makes it difficult to manage. Modern cryptography solves this problem by using a key, where the security of the cryptography depends on the secret of the key. Following diagram shows the encryption and decryption model with keys.

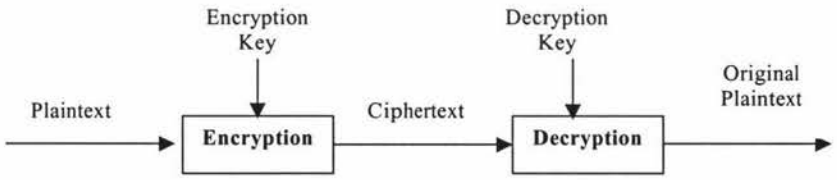


Figure 1.4: Encryption and Decryption with a key

There are two general categories of encryption algorithms: *conventional encryption* and *public-key encryption*.

In conventional encryption, the encryption and decryption use the same key. One of the most important encryption algorithms in the history of modern cryptography is the Data Encryption Standard (DES) algorithm. DES is a conventional block encryption algorithm using a 56 bits length key. It was adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as a standard cryptography algorithm. DES has been a worldwide standard for over 20 years, and it is nearly at the end of its life.

Public-key encryption is the greatest and perhaps the only true revolution in the entire history of modern cryptography. The concept of public-key encryption was invented by Whitfield Diffie and Martin Hellman in 1976 (Diffie & Hellman, 1976). In a public key encryption algorithm, encryption and decryption use different keys, one key is called the *public key* and it can be known to the public (that is why this kind of encryption algorithm is called public-key encryption). The public key is used by the party who want to encrypt the information, but any party who only knows the public key can not decrypt the ciphertext. The other key is the *private key,* and it is used to decrypt the ciphertext encrypted by the corresponding public key. The public key and private key should be used as a key pair, only the correct private key can decrypt the ciphertext encrypted by its corresponding public key. The purpose of a public-key cryptography algorithm is to hide the information in the plaintext. That is to say, the opponent parties can have access to the ciphertext and public key as well, but for the opponent, it should be very difficult to guess the contents of plaintext and private key by studying the ciphertext, public key and encryption algorithm.

1.4.4 Security of Encryption Algorithm and Cryptanalysis

The encryption algorithm should be strong enough to withstand the attack from cryptanalysts. The security degree of a cryptographic algorithm depends on how hard it is to break it. If the cost to break the algorithm is greater than the value of the encrypted data, then the algorithm is possibly strong enough. If the time to break the algorithm is

longer than the encrypted data must remains secret, then the algorithm is possibly strong enough. If the amount of data encrypted by a key is less than the required data to break the algorithm, then the system using this algorithm is possibly safe. However, there is always a chance of new breakthroughs in cryptanalysis. Therefore, none of the cryptographic algorithm is ever absolutely secure. To study security, one should also study the attack on the security, therefore some aspects of attack on encryption algorithms are discussed below.

According to Lars Knudsen (Knudsen, 1994), breaking an algorithm means:

(1) *Total break.* A cryptanalyst finds the key. Therefore the cryptanalyst can decrypt all the ciphertext using this key.

(2) *Global deduction.* A cryptanalyst finds an alternate algorithm without knowing the key. The alternate algorithm is equivalent to the decryption operation. In this way, the cryptanalyst can also get the plaintext of corresponding ciphertext.

(3) *Instance (or local) deduction.* A cryptanalyst finds the plaintext of an intercepted ciphertext.

(4) *Information deduction.* A cryptanalyst gains some information about the key or plaintext. This information could be a few bits of the key, some information about the form of the plaintext, and so forth. This information may be used in the future to break the algorithm.

In the cryptanalysis of an algorithm, it is always correct to assume that a cryptanalyst has knowledge of the encryption algorithm. Apart from the algorithm, the cryptanalyst needs to know some information about the ciphertext and plaintext before he/she will be able to break the algorithm. According to what the cryptanalyst known, the attack can be classified as:

(1) *Ciphertext only attack.* The cryptanalyst has got some ciphertext encrypted by the algorithm using the same key. The task of cryptanalyst is to recover as much as possible the plaintext, or even to deduce the key (*Total break* of the algorithm) in order to decrypt other ciphertext.

(2) *Known plaintext attack.* The cryptanalyst has got some piece of the ciphertext. Apart from that, the cryptanalyst also has access to one or more plaintext-ciphertext pairs

formed with the same key. The cryptanalyst's job is to find out the key used to encrypt the message.

(3) *Chosen plaintext attack.* The cryptanalyst has got some ciphertext. Apart from that, the cryptanalyst can also chose the plaintext to be encrypted and its corresponding ciphertext formed with the same key. The purpose of cryptanalyst is to find out the key.

(4) *Chosen ciphertext attack.* The cryptanalyst has got some ciphertext. Apart from that, the cryptanalyst can also chose purported ciphertext, together with its corresponding decrypted plaintext. The purpose of cryptanalyst is to deduce the key.

(5) *Chosen text attack.* The cryptanalyst has got some ciphertext to be decoded. Apart from that, the cryptanalyst can also chose a plaintext message together with its corresponding ciphertext, as well as chose purported ciphertext together with its corresponding decrypted plaintext. The purpose of cryptanalyst is to deduce the key

There are at least two other types of cryptanalytic attack:

(6) *Chosen-key attack.* The cryptanalyst has got some ciphertext to be decoded. Apart from that the cryptanalyst has some knowledge about the relationship between different keys. The purpose of the cryptanalyst is to find out the key. Some recent developed cryptanalysis deploys the weakness on key-schedule using chosen-key attack.

(7) *Rubber-hose cryptanalysis attack.* This is an attack on key management. The cryptanalyst get the key by using threats, blackmail, or torture someone or by bribery. These are all very powerful attacks and are often the best way to break an algorithm.

Most of the current conventional encryption algorithms are designed to withstand a *known-plaintext attack.* Only relatively weak algorithms fail to withstand a *ciphertext-only attack.*

All the cryptographic algorithms are breakable in a ciphertext-only attack, simply by exhaustively trying every possible key until a meaningful plaintext is obtained. This is called a *brute-force attack.* The number of operations required to break an algorithm by

brute-force attack depends on the length of a key. A key length of 56 bit of DES is too short to withstand a brute force attack by today's powerful computer (Hellman, 1976).

Cryptanalysis is a fast-moving field. Recent advances in cryptanalysis have revealed many weaknesses in some of the most recently developed cryptographic algorithms. One of the most significant advances in cryptanalysis in recent years is *differential cryptanalysis* (Biham & Shamir, 1990, 1992). Other recent developed cryptanalysis methods include *Linear Cryptanalysis* (Matsui, 1993), *Related-key cryptanalysis* (Biham, 1994), *differential-linear cryptanalysis* (Langford & Hellman, 1994).

1.5 Project Definition - Security Consideration of AudioGraph Multimedia Teaching Tools

The AudioGraph is a project developed in the Institute of Information Sciences and Technology of Massey University as copyrighted software tools. The AudioGraph courseware files produced by the AudioGraph Recorder may also be copyrighted. In order to protect the products, there should be some mechanism for preventing an unauthorized access of the courseware files as well as the software tools. This section discusses the security requirement and its associated aspects.

1.5.1 Security Requirement of AudioGraph

To protect the AudioGraph tools and courseware files, it is required that only the authorized user may be allowed to use the AudioGraph tools and access the courseware materials produced by AudioGraph Recorder. The courseware materials are on a CD-ROM or on a Web Server. It should not be easy for the unauthorized user to attack the system in order to access or make a copy of them.

For a courseware file (an .aep file) on a CD-ROM or on the Web Server, there must be some kind of information to mark the file, we call this marking information a "Copy Protection record" in the AudioGraph file. When a user wants to playback the course material in the .aep files via a AudioGraph Player (i.e., a PC Plug-in for the web browser), he/she must provide an identity called an "userId" and a correct password that

corresponds to the userId, otherwise he/she will not be allowed to play it back and the AudioGraph Player will exit execution. Figure 1.5 demonstrates this procedure.

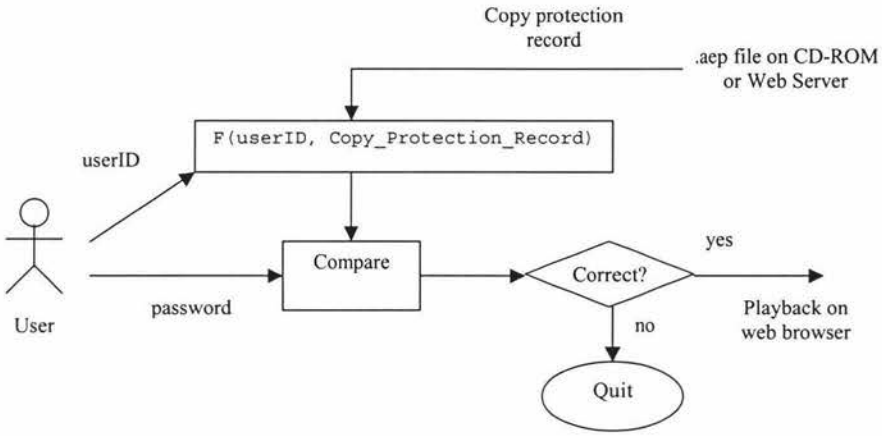


Figure 1.5: Usage Control of .aep file

In above diagram, 'F' is a one-way function, the output of the function is:

```
F(userId, Copy_Protection_Record)
```

The system will compare the output with user's input password, and see if:

```
F(userId,  Copy_Protection_Record) = password
```

If they are equal, then the user is allowed to playback the presentations in courseware file, otherwise, the user is not allowed to playback the presentations and the system will quit.

1.5.2 Copy Protection Record in the AudioGraph Files (.aep files)

The Copy Protection record is embedded in the AudioGraph files (.aep files), it is inserted into the AudioGraph files by a specific tool or by AudioGraph Recorder when it generated the course materials. The Recorder does not need to know about the userId, it must however, generate a unique key for a given unit of presentation, whatever the user decides this may be. The Copy Protection record includes three keys:

Company Key: 4-bytes, to give a unique identity of the company who produces the AudioGraph courseware product

Lower protection key: 4-bytes, used as a protection key

Upper protection key: 4-bytes, used as a protection key.

On encountering this record, the PC Plug-in without the security system embedded in it will terminate with a message asking the user to contact the supplier for a correct Player Plug-in version. For the PC Plug-in with the security system embedded in it, it will ask user to input userId and password and do a security check.

1.5.3 Other consideration of AudioGraph Multimedia Teaching Tools and Courseware files

The protection of AudioGraph software and courseware files from illegal copying depends on the security of the Web Server as well as the administration of the AudioGraph materials, including source code of the software, the Tools and courseware materials.

For the Web Server, it should withstand any kind of attack from intruders. An intruder may try to copy the source code of any AudioGraph materials and study them, and try to make use of them, he/she may possibly try to find out weakness of the AudioGraph security system and learn how to completely attack on them. The Web Server security is beyond the scope of this thesis. Interesting reader can refer to books about Web Security (Garfinkel, 1997; Stein, 1998).

Another important aspect of AudioGraph security is the administration of the AudioGraph materials. No matter how secure the security system and Web server are, if the management of user name and password file has a security bug, then the whole security system is useless. For instance, if the intruder gets access to a file storing all the user name and password pairs due to bad management, then the intruder can access and pass all the security checks which need the user name and password. In another situation, if an intruder is an expert on security programming and gets access to the source code of AudioGraph due to bad management of source code, he/she can study the source code and find out weakness of the security system, or even write his/her own tool such as a variant of PC Plug-in to bypass the security check, and that will make our security system worthless.

1.6 Summary

This chapter has covered the background area concerned with the Security System of AudioGraph project and given the project definition. The PNG image compression and GSM sound compression algorithms have been discussed. The basic concepts concerned with computer security and cryptography have been described. In the following chapter some of the popular used encryption algorithms will be discussed in more detail.

Chapter 2: Cryptography

The aim of this chapter is to give an introduction to the field of modern cryptography algorithms. Modern cryptography algorithms can be classified into two main categories: conventional encryption (also referred as symmetric encryption, secret-key or single-key encryption) and public-key encryption (also referred as asymmetric encryption or two-key encryption). This chapter describes some of the most widely used conventional encryption and public-key encryption algorithms.

2.1 Conventional Encryption Algorithms

Before the invention of public-key encryption, all the data encryption algorithms used in computer systems used conventional encryption. According to how the algorithms are constructed, encryption algorithms may be classified as *block cipher* and *stream cipher* algorithms. The block cipher treats the data to be encrypted as collection of blocks, the algorithm converts a block of plaintext into a block of ciphertext, it is widely used to encrypt static data such as files, databases, etc. The stream cipher treats the data as a stream of bits, it converts the plaintext bit-stream into ciphertext bit-stream, the Linear-Feedback Shift Register (LFSR) is an example of bit-stream cipher. Stream Cipher algorithms are widely used to encrypt the real-time data stream over a network, it has been widely used in military electronic communication system in recent history.

There are many block cipher and stream cipher algorithms. This section introduces some of the modern block cipher algorithms.

2.1.1 Data Encryption Standard (DES)

The Data Encryption Standard (DES) algorithm is the first encryption standard and the first widely used encryption algorithm. The DES was based on an Encryption Algorithm called Lucifer (Smith, 1971), which was first developed by a group of researchers in IBM. On May 15 of 1973, the NBS (National Bureau of Standards), now the NIST (National Institute of Standards and Technology) issued a public request for proposals

for a standard cryptographic algorithm. The Lucifer algorithm was the only promising candidate encryption algorithm they received. NBS made some modification to the algorithm, i.e., changed the key length from 128 bits to 56 bits and replaced the S-Box of the algorithm (NBS did not explained the reason to the public, therefore there has been much speculation on the key length, number of iterations and design of the S-boxes), then the encryption algorithm became the Data Encryption Algorithm (DES). Since then, DES has been widely used for many applications within the government, commercial company and other private section.

The Data Encryption Standard algorithm is a block cipher that transforms 64-bit data blocks using a 56-bit secret key, by means of permutation and substitution. It is officially described in FIPS PUB 46-2 (National Institute of Standards and Technology, 1993). The overall scheme for DES encryption is illustrated in Figure 2.1. A single iteration is illustrated in Figure 2.2.

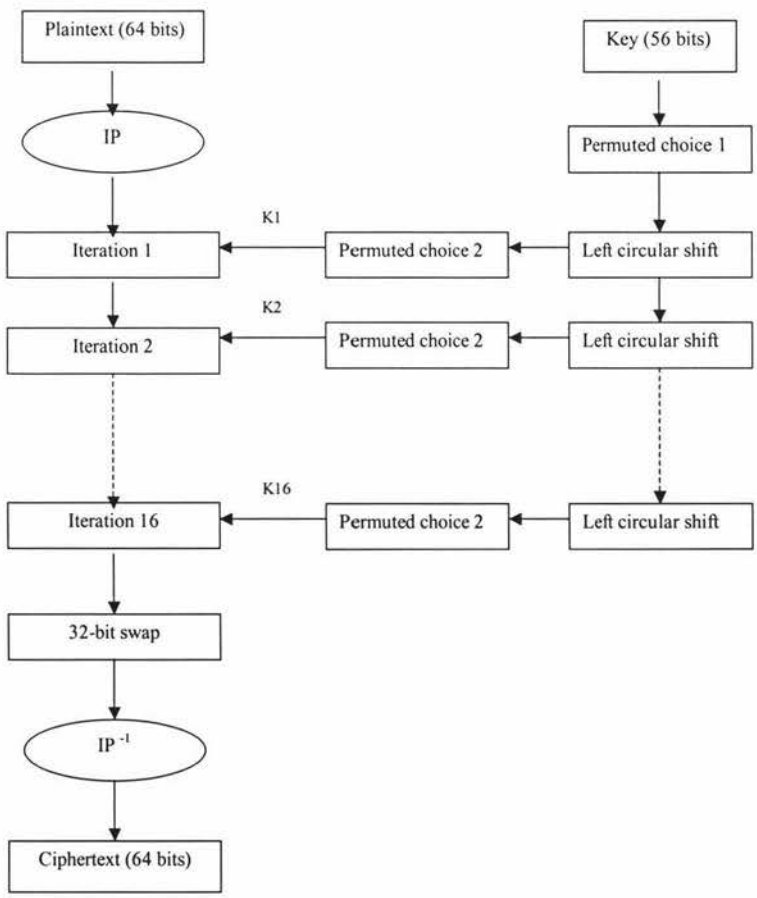


Figure 2.1: General Depiction of DES Algorithm

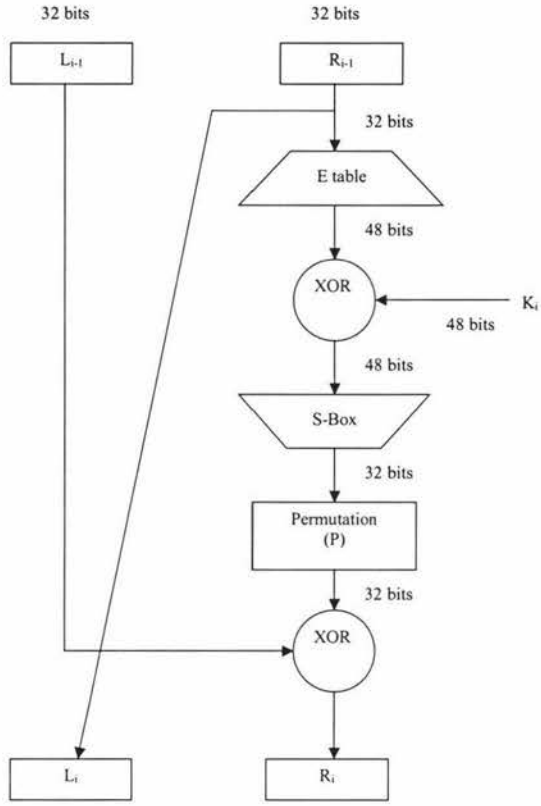


Figure 2.2: Single Iteration of DES Algorithm

There are two inputs to the DES algorithm: the 64 bits block plaintext and 56 bits key. The initial 64 bits key has been reduced to 56 bits DES key by ignoring every eighth bit, this bit can be used as parity bit in practical applications. The procedure to process the 64 bits block data is as following:

(1) Perform permutation to the 64 bits data block. We call it *Initial Permutation – IP*.

After the initial permutation, the following input 64 bits block:

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}
M_{17}	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}	M_{25}	M_{26}	M_{27}	M_{28}	M_{29}	M_{30}	M_{31}	M_{32}
M_{33}	M_{34}	M_{35}	M_{36}	M_{37}	M_{38}	M_{39}	M_{40}	M_{41}	M_{42}	M_{43}	M_{44}	M_{45}	M_{46}	M_{47}	M_{48}
M_{49}	M_{50}	M_{51}	M_{52}	M_{53}	M_{54}	M_{55}	M_{56}	M_{57}	M_{58}	M_{59}	M_{60}	M_{61}	M_{62}	M_{63}	M_{64}

Will become the following output bits block:

M_{58}	M_{50}	M_{42}	M_{34}	M_{26}	M_{18}	M_{10}	M_2	M_{60}	M_{52}	M_{44}	M_{36}	M_{28}	M_{20}	M_{12}	M_4
M_{62}	M_{54}	M_{46}	M_{38}	M_{30}	M_{22}	M_{14}	M_6	M_{64}	M_{56}	M_{48}	M_{40}	M_{32}	M_{24}	M_{16}	M_8
M_{57}	M_{49}	M_{41}	M_{33}	M_{25}	M_{17}	M_9	M_1	M_{59}	M_{51}	M_{43}	M_{35}	M_{27}	M_{19}	M_{11}	M_3
M_{61}	M_{53}	M_{45}	M_{37}	M_{29}	M_{21}	M_{13}	M_5	M_{63}	M_{55}	M_{47}	M_{39}	M_{31}	M_{23}	M_{15}	M_7

(2) After the initial permutation, the 64-bits block will be split into two halves: The left 32 bits block is called L[0], and the right 32 bits block is called R[0].

(3) Then the sixteen subkeys, k_1 k_2 ... k_{16}, are applied to the data block through sixteen similar operations, each operation is called an iteration, starting with iteration one. Figure 2.2 demonstrates the operations within one iteration.
Inside each iteration, the output of left 32 bits half will be the same as right half of 32 bits input, i.e., L[i] = R[i-1]. It will perform the following operation to the right 32 bits half to get the R[i]:

a. Expand the 32-bits R[i-1] into 48 bits according to pattern defines in Figure 2.3. This operation is called *Expansion Permutation*. The expansion table is sometimes called the *E-box* or *E-table*.
b. Exclusive-or with 48-bits length subkey K_i
c. Perform the *S-Box* substitution: break the 48 bits block into eight 6-bits block, let us say B[1] to B[8]. For each block B[i], take the 1st and 6th bits of B[i] together as a 2-bit value (call it m) which act as index into the row in S[i] to look in for the substitution. Then take the 2nd through 5th bits of B[i] together as a 4-bit value (call it n) which act as index into the column in S[i] to find the substitution. Then replace B[i] with S[i][m][n]. Table 2.1 is the S-Box.
d. Perform permutation, called it *P-Box* permutation. Refer to Table 2.2 for the permutation function.
e. Exclusive-or the resulting value with L[i-1].
The right 32-bits half of output can be present as:

$$R[i] = L[i-1] \oplus f(R[i-1], K_i)$$

In Conclusion, each iteration perform the following operation:

$L[i] = R[i-1]$

$R[i] = L[i-1] \oplus f(R[i-1], K_i)$

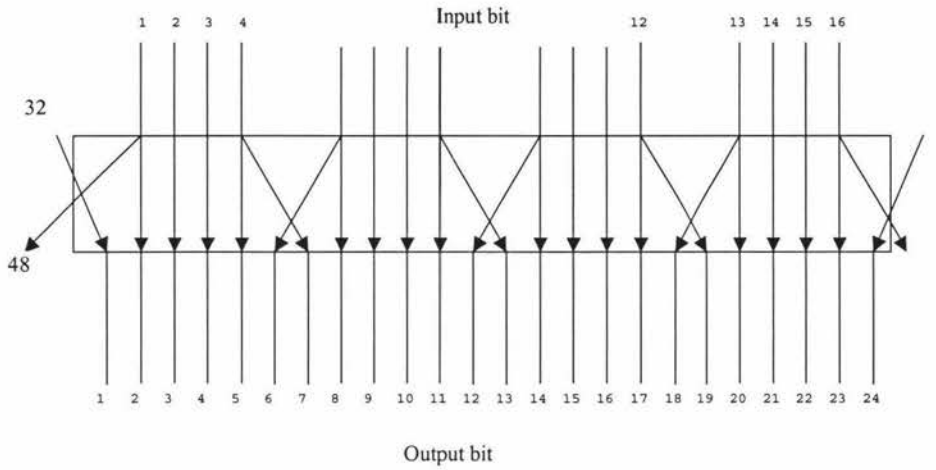


Figure 2.3: Expansion permutation (E-table)

S[1]

```
14  4 13  1  2 15 11  8  3 10  6 12  5  9  0  7
 0 15  7  4 14  2 13  1 10  6 12 11  9  5  3  8
 4  1 14  8 13  6  2 11 15 12  9  7  3 10  5  0
15 12  8  2  4  9  1  7  5 11  3 14 10  0  6 13
```

S[2]

```
15  1  8 14  6 11  3  4  9  7  2 13 12  0  5 10
 3 13  4  7 15  2  8 14 12  0  1 10  6  9 11  5
 0 14  7 11 10  4 13  1  5  8 12  6  9  3  2 15
13  8 10  1  3 15  4  2 11  6  7 12  0  5 14  9
```

S[3]

```
10  0  9 14  6  3 15  5  1 13 12  7 11  4  2  8
13  7  0  9  3  4  6 10  2  8  5 14 12 11 15  1
13  6  4  9  8 15  3  0 11  1  2 12  5 10 14  7
 1 10 13  0  6  9  8  7  4 15 14  3 11  5  2 12
```

S[4]

```
 7 13 14  3  0  6  9 10  1  2  8  5 11 12  4 15
13  8 11  5  6 15  0  3  4  7  2 12  1 10 14  9
10  6  9  0 12 11  7 13 15  1  3 14  5  2  8  4
 3 15  0  6 10  1 13  8  9  4  5 11 12  7  2 14
```

S[5]

```
 2 12  4  1  7 10 11  6  8  5  3 15 13  0 14  9
14 11  2 12  4  7 13  1  5  0 15 10  3  9  8  6
 4  2  1 11 10 13  7  8 15  9 12  5  6  3  0 14
11  8 12  7  1 14  2 13  6 15  0  9 10  4  5  3
```

S[6]

```
12  1 10 15  9  2  6  8  0 13  3  4 14  7  5 11
10 15  4  2  7 12  9  5  6  1 13 14  0 11  3  8
 9 14 15  5  2  8 12  3  7  0  4 10  1 13 11  6
 4  3  2 12  9  5 15 10 11 14  1  7  6  0  8 13
```

S[7]

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S[8]

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 2.1: S-Box

Output bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
From Input bit:	16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
Output bit	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
From Input bit:	2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Table 2.2: P-Box Permutation

(4) After the 16th iteration, the data block will perform a permutation, this permutation function is the reverse of the Initial Permutation function. We call it Inverse Initial Permutation (IP^{-1}).Then the output of Inverse Initial Permutation is ciphertext.

The procedure required to produce the sub-key is called the *key schedule.* The sixteen 48-bits sub-keys of DES are calculated from the 56-bits key by the following step:

(1) Perform the permutation on the 56-bits key according following table, it is called Permuted Choice One (PC-1).

Output bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14
From Input bit:	57	49	41	33	25	17	9	1	58	50	42	34	26	18
Output bit	15	16	17	18	19	20	21	22	23	24	25	26	27	28
From Input bit:	10	2	59	51	43	35	27	19	11	3	60	52	44	36
Output bit	29	30	31	32	33	34	35	36	37	38	39	40	41	42
From Input bit:	63	55	47	39	31	23	15	7	62	54	46	38	30	22
Output bit	43	44	45	46	47	48	49	50	51	52	53	54	55	56
From Input bit:	14	6	61	53	45	37	29	21	13	5	28	20	12	4

Table 2.3: Permuted Choice One (PC-1)

(2) Split the permuted key into two halves. The first 28 bits are called C[0] and the last 28 bits are called D[0].

(3) Perform one or two circular left shifts on both C[i-1] and D[i-1] to get C[i] and D[i], respectively. The number of shifts per iteration are given in the table below.

Iteration number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shifts number	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Permute the concatenation C[i]D[i] as indicated in the table 2.4. This will yield the 48-bits long sub-key for iteration i.

Output bit	:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
From Input bit	:	14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4

Output bit	:	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
From Input bit	:	26	8	16	7	27	20	13	2	41	52	31	37	47	55	30	40

Output bit	:	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
From Input bit	:	51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

Table 2.4: Permuted Choice One (PC-2)

The Data Encryption Standard is important in modern data encryption, not only because it is the first data encryption algorithm, but also because it gave us the basic building block of modern conventional encryption algorithm. Some of the important concepts of block encryption algorithms derived from DES are:

- ***Feistel Network***: The DES used a production system called *Feistel Network.* The idea of Feistel Network is to take the block of length n and divide it into two halves of length n/2: L and R. Of course, n must be even. Then these two halves go through a number of times of round function. The output of round *i* can be defined as:

  $$L_i = R_{i-1}$$

  $$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

  K_i is the subkey used in the *ith* round and *f* is an arbitrary round function. No matter what the round function is, the *Feistel Network* guarantees that the decryption operation is the reverse of the encryption operation. This can be seen when we rearrange the above terms as:

$$R_{i-1} = L_i$$

$$L_{i-1} \oplus f(R_{i-1}, K_i) \oplus f(R_{i-1}, K_i) = R_i \oplus f(R_{i-1}, K_i)$$

$$\Rightarrow L_{i-1} = R_i \oplus f(R_{i-1}, Ki) = R_i \oplus f(L_i, K_i)$$

This means a cipher that uses this construction is guaranteed to be invertible (because exclusive-or operation is invertible), no matter what the round function *f* is, and *f* does not need to be invertible. Therefore, the designer of the encryption algorithm can design *f* to be as complicated as possible, the decryption algorithm is just the reverse procedure of the encryption algorithm.

The Feistel Network structure can be found on many of block encryption algorithm, such as Lucifer (Smith, 1971), FEAL (Shimizu & Miyaguchi, 1987), GOST (Schneier, 1995), CAST (Adams, 1997), Blowfish (Schneier, 1994; Counterpane Internet Security Inc., 2000) and many others.

- ***S-Box***: According to Shannon's *Information Theory* (Shannon, 1949), there are two basic techniques for obscuring the redundancies in a plaintext message – *Confusion* and *Diffusion*. In the practice of cryptography, confusion is achieved by a substitution table. Confusion serves to hide any relationship between the plaintext, the ciphertext and the key. The S-Boxs in DES perform confusion, it is the only nonlinear operation in the encryption algorithm. The diffusion spreads the influence of individual plaintext or key bits over as much of the ciphertext as possible. It will hide the statistical relationships and makes cryptanalysis more difficult. Diffusion is a linear operation, in practice it is achieved by permutation. In DES algorithm, diffusion is archived through P-box and Permutation.

  The concept of S-Box is important in the design of a block cipher algorithm. The strength of various Feistel Networks is tied directly to their S-boxes. There has been a lot of research undertaken on the S-boxes of DES. This has revealed that the S-boxes of DES are optimized against *differential cryptanalysis* (Biham & Shamir, 1990, 1992), although the DES became standard before the *differential cryptanalysis* was known by the world. However, the S-boxes are not optimized against *linear cryptanalysis* (Matsui, 1993).

In a data security application, the DES algorithm is a basic building block. DES can be used in four different operation mode (Linn, 1989): *Electronic Codebook (ECB) Mode*,

Cipher Block Chaining (CBC) Mode, *Cipher Feedback (CFB) Mode* and *Output Feedback (OFB) Mode*.

Before DES, there was no standard encryption algorithm, so cryptanalysis did not have a target focus. Now DES has been the data encryption standard for more than 20 years, on the other hand it has also been the target of a great deal of cryptanalysis for more than 20 years. Furthermore, in past three decades, the IC computing power has increased by 1,000,000,000,000 times. Both issues make the DES insecure. Actually there has been much progress on the cryptanalysis of the DES algorithm. Today, the world record for breaking DES is 22 hours and 15 minutes. With a specially designed supercomputer called "DES Cracker" in the Electronic Frontier Foundation (EFF), and a worldwide network of nearly 100,000 PCs on the Internet, a worldwide coalition of computer enthusiasts called *Distributed.Net* won the RSA Code-Breaking Contest of breaking the DES in only 22 hours and 15 minutes (The Electronic Frontier Foundation, 1999). This shows that 56-bit keys encryption algorithm such as DES have nearly reached the end of their life. New encryption algorithms with longer key are required. Before a new standard encryption algorithm was developed and adopted, NIST recommended the use of Triple DES (National Institute of Standards and Technology, 1999b) instead of DES.

2.1.2 IDEA

IDEA is the most successful encryption algorithm that was designed to replace the DES algorithm. It was designed by Xuejia Lai and James Massey in 1990 and the first algorithm was called PES (Proposed Encryption Standard). Later the author strengthened the PES against *differential cryptanalysis* and called the new algorithm IPES (Improved Proposed Encryption Standard), and then the author changed its name to IDEA (International Data Encryption Algorithm).

IDEA is based on some impressive theoretical foundations. Although cryptanalysis has made some progress in attacking reduced-round variants of IDEA, and some weak keys of IDEA have been found by cryptanalysis, IDEA seems to remain strong against the currently known cryptanalysis. Unlike other block cipher algorithms, IDEA is patented by the Swiss firm called Ascom, although it is free for noncommercial use. IDEA is best

known to the world as a block cipher used in the popular encryption system called Pretty Good Privacy (PGP).

Unlike DES, IDEA uses a 128-bit key. The key is used to produce 52 16-bits sub-keys. The algorithm is not based on a Feistel Network. It uses three operations:

- Bit-by-bit Exclusive-OR, denoted as 
- Addition modulo 2^{16}, denoted as 
- Multiplication modulo $2^{16}+1$, denoted as 

The IDEA algorithm is elegant and simple. The standard IDEA is an 8-iteration algorithm, each round use 6 sub-keys. After the 8th iteration, it will perform output transforming which uses 4 sub-keys.

The procedure of each round is clearly illustrated by Figure 2.4.

The sub-key generation of IDEA is also elegant and simple. The first eight subkeys: z_1, z_2, …, z_8 are taken directly from the 128 bit key of IDEA: z_1 being equal to the first 16 bits of key, z_2 being the second 16 bits of key, and so on. Then the key will be circular shifted left by 25 bits, and the next 8 subkeys are extracted in the same way. This procedure is repeated until all the 52 subkeys are generated.

From above description, it is obvious that IDEA is simpler and more elegant than DES, and the speed of current software implementation of IDEA is similar to DES (RSA Security Inc., 2002). Most importantly, IDEA is more secure than DES, so IDEA is a better encryption algorithm than DES. The only drawback of IDEA is that it is patented (Massey & Lai, 1991, 1993) for commercial use. It is patented in Europe and the United States. The patent is held by Ascom-Tech AG in Geverbepark of Switzerland.

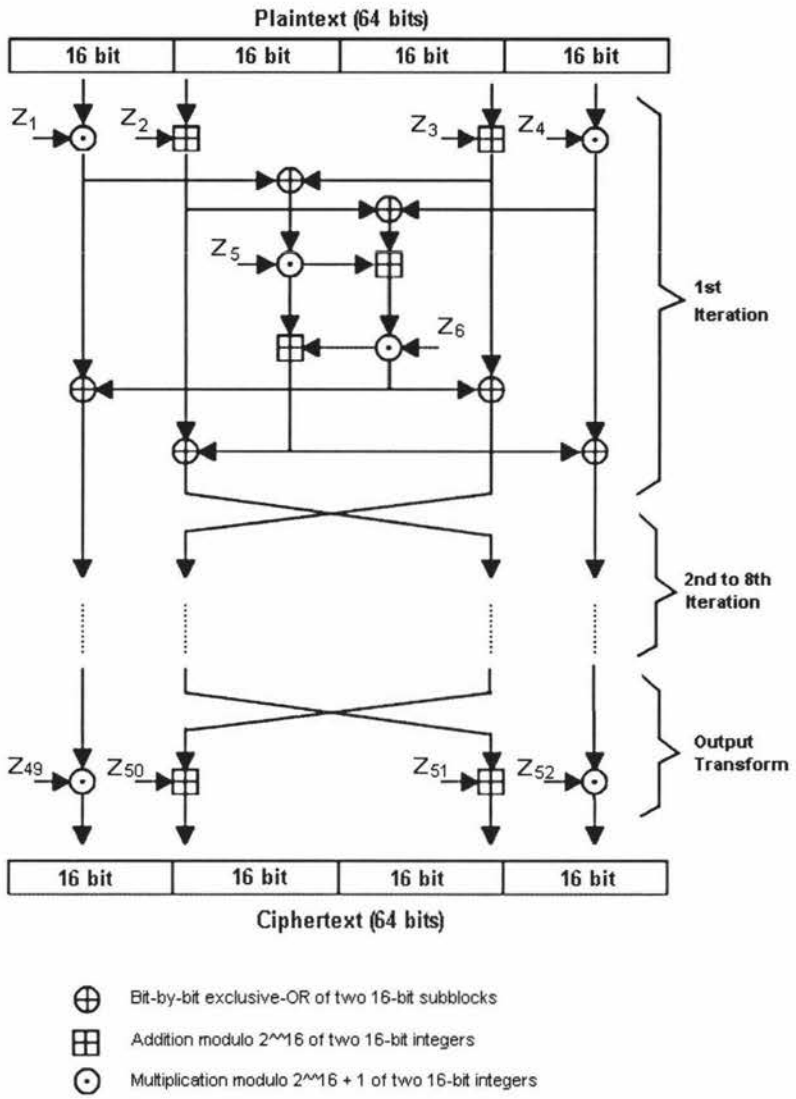


Figure 2.4: IDEA encryption algorithm

2.1.3 AES – Rijndael

Since DES algorithm has become obsolete, a replacement algorithm is required. Before the next encryption algorithm standard is adopted, triple DES has been endorsed by NIST as a temporary standard to be used.

On January 2, 1997, NIST announced the initiation of the AES development effort (National Institute of Standards and Technology, 1997a) and made a formal call for algorithms (National Institute of Standards and Technology, 1997b) on September 12, 1997. The call stipulated that the AES would specify an unclassified, publicly disclosed

encryption algorithm(s), available royalty-free, worldwide. In addition, the algorithm(s) must implement symmetric key cryptography as a block cipher and (at a minimum) support block length of 128-bits and key length of 128-, 192-, and 256-bits.

On August 20, 1998, NIST announced a group of fifteen AES candidate algorithms (National Institute of Standards and Technology, 1998) at the First AES Candidate Conference (AES1). A Second AES Candidate Conference (AES2) was held in March 1999 to discuss the results of the analysis on the candidate algorithms. Using the analyses and comments received, NIST selected five algorithms from the fifteen as finalists (National Institute of Standards and Technology, 1999a), they are **MARS** (IBM Research, 2001), **RC6** (RSA Security Inc., 2001), **Rijndael** (Rijmen, 1999; National Institute of Standards and Technology, 2001), **Serpent** (Anderson, 2001), **and Twofish** (Counterpane Internet Security Inc., 2001). These finalist algorithms received further analysis during a more in-depth review period prior to the selection of the final algorithm(s) for the AES. The third AES Conference (AES3) was held April 13-14, 2000 in New York, USA nearly at the end of Round 2 public analysis period. After the close of the Round 2 public analysis period on May 15, 2000, NIST studied all available information in order to make a selection for the AES. At the last AES conference, **Rijndael** got 86 votes, **Serpent** got 59 votes, **Twofish** 31 votes, **RC6** 23 votes and **MARS** 13 votes. Therefore on October 2, 2000, NIST announced that it has selected **Rijndael** as a proposal for the AES. Now the Rijndael has become official standard and the interested reader can refer to the Draft Federal Information Processing Standard (FIPS) for the AES (National Institute of Standards and Technology, 2001).

The Rijndael block cipher was designed by Joan Daemen and Vincent Rijmen in Belgium, its design was strongly influenced by the design of another block cipher of the same authors: **Square** (Daemen et al., 1997).

The key length of Rijndael is defined to be either 128, 192 or 256 bits. Rijndael has a variable block length of either 128, 192 or 256 bits. All nine combinations of key length and block length are possible. Unlike other block cipher algorithm, the number of iteration is 10, 12 or 14, depends on the block size and key length.

Rijndael is an iterated block cipher, the number of round depends on the block length and Cipher key length. Unlike DES and many other block ciphers, Rijndael is not a Feistel Network structure. The operations in Rijndael are defined at byte level, with bytes representing elements in the finite field (Beresford, 2002). In Rijndael, the intermediate cipher result is called the *State*.

Both State and Cipher Key in Rijndael can be pictured as rectangle array with four rows, with each element inside the array is a byte. The number of columns in a State is denoted by *Nb*, it equals to the block length divided by 32. The number of columns in a Cipher Key is denoted by *Nk*, it equals to the key length divided by 32. Figure 2.5 is an example of State and Cipher Key.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

Example of State (Nb = 6)

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Example of Cipher Key (Nk =

Figure 2.5: Example of State and Cipher Key layout

Suppose the input bytes order is b_0, b_1, b_2, b_3, b_4, b_5, …, b_{22}, b_{23}, they will be mapped onto the state bytes in the order of $a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, …, $a_{2,5}$, $a_{3,5}$. The output ciphertext is extracted from the last state by taking the state bytes in the same order. The Cipher Key bytes are mapped onto the Cipher Key array in the same way, i.e., the byte order of Cipher Key would be $k_{0,0}$, $k_{1,0}$, $k_{2,0}$, $k_{3,0}$, $k_{0,1}$, $k_{1,1}$, $k_{2,1}$, $k_{3,1}$, $k_{4,1}$, ….

The number of rounds is denoted by *Nr*. The relation between *Nb*, *Nk* and *Nr* is given in Table 2.5.

Table 2.5: Number of rounds (Nr) as a function of the block and key length

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

The Rijndael cipher

According to official Rijndael specification submitted for the AES by the algorithm's designers (Daemen & Rijmen, 1999), the algorithm can be depicted in pseudo C notation as:

```
Rijndael(state, CipherKey)
{
      KeyExpansion(CipherKey, ExpandedKey) ;
      AddRoundKey(State, ExpandedKey) ;
      For(i=0; i<Nr; i++)
            Round(State, ExpandedKey + Nb * i) ;
      FinalRound(State, ExpandedKey + Nb * Nr) ;
}
```

In above algorithm, the *KeyExpansion(CipherKey, ExpandedKey)* will expand the Cipher Key into Round Key denoted by ExpandedKey. The *AddRoundKey(State, ExpandedKey)* will add the first round key to the State. Then it will perform round transformation for *Nr* times. After perform the final round transform: *FinalRound(State, ExpandedKey + Nb*Nr)*, the output Ciphertext will be extract from the State.

The Round Transformation

The transformation function of each round (except the final round) can be depicted in pseudo C notation (Daemen & Rijmen, 1999) as:

```
Round(State, RoundKey)
{
      ByteSub(State);
      ShiftRow(State);
      MixColumn(State);
      AddRoundKey(State, RoundKey);
}
```

The final round is slightly different, it is defined by:

```
FinalRound(State, RoundKey)
{
      ByteSub(State);
      ShiftRow(State);
      AddRoundKey(State, RoundKey);
}
```

Here, "RoundKey" is a Round Key for this round. The Round Keys are derived from the Cipher Key by Key Schedule method of Rijndael algorithm.

- ***Addition*** and ***Multiplication*** in Rijndael

  Some operations in Rijndael are defined at byte level, with bytes representing elements in the Finite Field GF(2^8). In Rijndael, the addition operation correspondence with the simple bitwise XOR (denoted by $\oplus$) is at the byte level. The multiplication operation in Rijndael is performed by multiplication of the polynomials modulo an irreducible polynomial of degree 8. This irreducible polynomial is called *m(x)* and given by

  $$m(x) = x^8 + x^4 + x^3 + x + 1$$

  or '11B' in hexadecimal representation, or 283 in decimal representation. This number is a prime number, so its corresponding polynomial is irreducible.

  Other operations in Rijndael are defined in terms of 4-byte *words*.

- ***ByteSub(State)***

  In the Round Transformation, the "*ByteSub(State)*" transformation is a non-linear byte substitution, operating on each of the State bytes independently. This operation can be viewed as 8 × 8 S-Box operation. Each byte in the *State* will be

substituted by another byte according to the S-Box. For each byte in the State, following operations are performed:

(1) Calculate the multiplication inverse in GF(2^8) of the byte value.

(2) Applying the following transformation to the byte:

$$\begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \\ y4 \\ y5 \\ y6 \\ y7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Here, vector $\begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \end{bmatrix}$ is the input byte, vector $\begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \\ y4 \\ y5 \\ y6 \\ y7 \end{bmatrix}$ is the output byte

- ***ShiftRow(State)***

In the "*ShiftRow(State)*" operation, each row of the State are cyclically shifted over different offsets. Row 0 is not shifted, Row 1 is shifted left over C1 bytes, row 2 over C2 bytes and row 3 over C3 bytes. The value of C1, C2 and C3 depend on the block length, they are specified by Table 2.6.

Table 2.6: Shift offsets for different block lengths (in byte)

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

- ***MixColumn(State)***

  The "*MixColumn(State)*" transformation will perform on each column of the *State*. The columns of the State are considered as polynomials over GF(28), with each element byte in the column as coefficients. Then the polynomial will multiply modulo (x^4+1) with the fixed polynomial c(x), given by

  $$c(x) = '03'x^3 + '01'x^2 + '01'\ x + '02'$$

  The result can be written as matrix multiplication below (for column j):

  $$\begin{bmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0j} \\ a_{1j} \\ a_{2j} \\ a_{3j} \end{bmatrix} \qquad j = 0, 1, \ldots, Nb$$

  When calculation above matrix multiplication, the addition operation is simple bitwise XOR, the multiplication is polynomial multiplication module m(x):

  $$m(x) = x^8 + x^4 + x^3 + x + 1$$

  After this operation, the columns are mixed, i.e., each element in a column is the mixture of other elements inside this comunn.

- ***AddRoundKey(State, RoundKey)***

  In the last round, this operation is omitted. In this "*AddRoundKey(State, RoundKey)*" operation, a Round Key is applied to the *State* by a simple bitwise XOR operation of each element (byte) in the *State* and *RoundKey*. The Round Key is derived from the Cipher Key by key schedule method of this Rijndael encryption. The size of Round Key is the same as the size of the *State*.

Key Schedule

The Cipher Key is the key input into the Rijndael algorithm. An example of a cipher key can be found in Figure 2.5. The Cipher Key should be used to derive the Round Key, which will be used in the round transformation. This is by means of the key schedule. The key schedule of Rijndael consists of two components: the Key Expansion and the Round Key Selection.

- *Key Expansion*

The Expanded Key is a linear array of 4-byte words and is denoted by ExpandedKey[Nb * (Nr + 1)]. There are two different expansion function, one is for $Nk \leq 6$, another is for $Nk > 6$. The expansion functions are described below.

If $Nk \leq 6$, we have (Daemen & Rijmen, 1999):

```
KeyExpansion(byte CipherKey[4*Nk], word ExpandedKey[Nb*(Nr+1)])
{
      for(i = 0; i < Nk; i++)
      ExpandedKey[i] = (CipherKey[4*i], CipherKey[4*i+1],
                        CipherKey[4*i+2], CipherKey[4*i+3]);

      for(i = Nk; i < Nb * (Nr + 1); i++)
      {
            temp = ExpandedKey[i - 1];
            if (i % Nk == 0)
                  temp = SubByte(RotByte(temp)) ^ Rcon[i/Nk];
            ExpandedKey[i] = ExpandedKey[i - Nk] ^ temp;
      }
}
```

If $Nk > 6$, we have (Daemen & Rijmen, 1999):

```
KeyExpansion(byte CipherKey[4*Nk], word ExpandedKey[Nb*(Nr+1)])
{
      for(i = 0; i < Nk; i++)
      ExpandedKey[i] = (CipherKey[4*i], CipherKey[4*i+1],
                        CipherKey[4*i+2], CipherKey[4*i+3]);

      for(i = Nk; i < Nb * (Nr + 1); i++)
      {
            temp = ExpandedKey[i - 1];
            if (i % Nk == 0)
                  temp = SubByte(RotByte(temp)) ^ Rcon[i/Nk];
            else if (i % Nk == 4)
                  temp = SubByte(RotByte(temp));
            ExpandedKey[i] = ExpandedKey[i - Nk] ^ temp;
      }
}
```

Here, Rcon[i/Nk] is a round constant, it can be defined by:

Rcon[i] = (RC[i], '00', '00', '00'),

RC[i] is an element in GF(2^8) with a value of $x^{(i-1)}$.

- *Round Key Selection*

  Round Key selection is simple, the round key i is given by the expanded key buffer words:

  ExpandedKey[Nb*i] to ExpandedKey[Nb*(i+1)].

The Rijndael cipher has the strength to stand against all known attacks. There is no possibility for weak keys, semi-weak keys and equivalent keys in Rijndael. The Rijndael has been designed to against the *Differential Cryptanalysis* (Biham & Shamir, 1990, 1992), *Linear Cryptanalysis* (Matsui, 1993), *Truncated Differentials* attack, the *Square attack*, *Interpolation attacks* and *Related-key* attacks. According to the NIST's Report on the Development of the Advanced Encryption Standard (AES) (Nechvatal et al., 2000), no attacks have been reported against any of the finalists. The only attacks that have been reported are against simplified variant of the algorithms in the finalists. Although the security margin of Rijndael is on the low side among the finalists (SERPENT appears to be the most secure algorithm among the finalists), Rijndael appears to be the best among the finalists.

After Rijndael become the official standard, it has the potential to be used by millions of people. NIST anticipates that this algorithm will be used widely in government and commercial application – both domestically and internationally. It is estimated that the AES has the potential to remain secure well beyond twenty years.

2.1.4 Other block cipher algorithms

Apart from the block cipher algorithms mentioned above, there are many other block cipher algorithms around the world for varying purpose. Some of the algorithms were designed by a commercial party and have commercial use, for example, RC2 and RC5 are products of RSA laboratories. Some of the algorithms have had popular use in different countries, for example, GOST is a similar standard to DES in former Soviet Union. Some of the algorithms are patented algorithm, for example, IDEA is patened in Europe (Massey & Lai, 1991) and United States (Massey & Lai, 1993). However most

of the block cipher algorithms are invented in academic field and only for research purpose.

2.2 Public-Key Encryption Algorithms

The concept of public-key cryptography was introduced by Whitfield Diffie and Martin Hellman in 1976 (Diffie & Hellman, 1976).

Public-Key encryption algorithm is different from conventional encryption algorithm. The contribution of Public-Key algorithm is the notation that keys could come in pairs – a public key used in encryption and a private key used in decryption, it should be infeasible to generate one key from the other. Since the invention of this notation, numerous public-key algorithms have been proposed. But many of them are insecure or impractical. Only a few of them are both practical and still secure. These algorithms are generally based on the hard problem of Complexity Theory.

Of those public-key algorithms that are both practical and secure, some are only suitable for key distribution, some are suitable for data encryption (of course they are also suitable for key distribution), some are suitable for digital signature. As far as the author knows, only three algorithms work well for both encryption and digital signature: RSA, ElGamal and Rabin.

From the security point of view, it is not to say that public-key algorithms are more secure than conventional algorithms. The speed of encryption and decryption is very slow compared with conventional encryption algorithms. So they are not usually used in data encryption and decryption. Instead public-key algorithms have been well been used in key distribution and in digital signatures.

In a conventional encryption algorithm, there is only one key, both the person who does encryption and the person who does decryption share the same key. This key is the most sensitive piece of information and there is always a risk to in exchanging this key. You have to make sure that only the 'right' persons knows this secret key. So there must be a secure channel to exchange the key. With the public-key algorithm, key exchange

becomes very easy. In the public-key cryptosystem, the public key can just be published. So anyone can encrypt messages, but only the person who owns the private key can decrypt it. With conventional encryption algorithm, a digital signature is impossible. The digital signature is another important application of public-key cryptography. The application of public-key cryptography to digital signatures can be explained by an example. Suppose a person called A wants to sign a piece of electronic document, he/she can sign it by encrypting this electronic document using his/her private key. If other person wants to verify this signature, they can decrypt this electronic document using A's public key, if the result is meaningful (that means the decryption is successful), then the signature is valid. However, there are two issues about digital signatures that must be considered: one is whether public-key cryptography is suitable for digital signature; another is that the public key should be certified, that is to say, it should have been certified by trusted third party that the public-key belongs to exactly the right person.

A public-key encryption scheme is vulnerable to a brute-force attack. The countermeasure to this attack is to use a longer key. However, there is tradeoff to be considered. Public-key systems use some sort of invertible mathematical function, the complexity of calculating these functions may not scale linearly with the number of bits in the key but grow more rapidly than that. Thus the key length must be large enough to make brute-force attack impractical but be small enough for practical encryption and decryption. In practice, all of the secure and practical public-key algorithms are too slow to encrypt and decrypt bulk data. They encrypt and decrypt data much more slowly than conventional algorithm. In practical cryptosystem, when it is required to encrypt and decrypt bulk data, the system uses a conventional algorithm with a random session key to encrypt the bulk data, and uses a public-key algorithm to encrypt the random session key and to distribute the key to the encrypted bulk-data receiver. In this way, the public-key algorithm is used to setup a secure channel for distribution of the random session key. But this approach has a possible security problem. Since the cryptanalyst has access to the public key, he/she can always choose any message to encrypt. If the message been encrypted is small, then the cryptanalyst can find out the message by exhaustive search. This problem can be solved by padding messages with a string of random bits, thus identical plaintext messages will be encrypted into different ciphertext messages.

The remaining part of this section introduces some of the important public-key encryption algorithm.

2.2.1 Knapsack Algorithm

Knapsack algorithm is the first public-key encryption algorithm invented by Ralph Merkle and Martin Hellman (Hellman, 1979; Merkle & Hellman, 1978). Knapsack algorithm gets its security from the knapsack problem.

The knapsack problem is simple. Given a cargo vector $\mathbf{a} = (a_1, a_2, a_3, \ldots, a_n)$ and a sum S, the task is to compute the values of vector $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$ so that:

$$S = \mathbf{a} \bullet \mathbf{x} = a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_nx_n$$

The value of x_i can be either zero or one. The cargo vector **a** can be considered as a list of potential elements to be put in the knapsack, with each vector element equal to the weight of the corresponding element. Vector **x** is considered to be a list of selected elements, with x_i=1 indicating element a_i is selected for inclusion in the knapsack.

For data encryption, cargo vector **a** is used as public key, vector **x** is used as plaintext and sum S is ciphertext. To do the encryption, one need only calculate the value of the sum S, given cargo vector **a** and plaintext **x**. To decrypt the ciphertext sum S, one will have to find out the vector **x** given sum S and cargo vector **a**. It is obvious very easy and quick to calculate sum S, even thought the number of elements in cargo vector **a** and vector **x** is very large, this means the calculation of encryption is easy and quick. However, it is not so easy and quick to find out the value of vector **x** given the sum S and cargo vector **a**. To find out vector x, one will have to test all possible solutions until you stumble on the correct one. That means, given ciphertext and public key, it is hard to guess the contents of the plaintext. So knapsack is a hard problem.

If we impose the condition that each element of vector **a** is larger than the sum of the preceding elements:

$$a_i > \sum_{j=1}^{i-1} aj \qquad 1 < i \leq n$$

Then vector **a** would be a *superincreasing vector*. In this case the resulting problem is easy to solve, we call it superincreasing knapsack. For example, consider a total knapsack weight of 69 and a cargo vector **a**:

a = (2, 3, 6, 13, 25, 51)

The largest weight, 51, is less than 69, so 51 is in the knapsack. Subtracting 51 from 69 leaves 18. The next weight, 25, is greater than 18, so 25 is not in the knapsack. The next weight, 13, is less than 18, so 13 is in the knapsack. Subtracting 13 from 18 leaves 5. The next weight, 6, is greater than 5, so 6 is not in knapsack. Continuing this process will show that both 2 and 3 are in the knapsack, and the total weight is brought to 0, so the solution has been found.

Non-superincreasing, or normal, knapsacks are hard problems. The knapsack algorithm developed by Ralph Merkle and Martin Hellman is based on the property of knapsack problem. Merkle and Hellman developed a technique for converting a superincreasing knapsack problem into a normal knapsack problem using modular arithmetic. The private key is a sequence of weights for a superincreasing knapsack problem. The public key is a sequence of weights for a normal knapsack problem with the same solution. So with a public key, the user can easily encrypt plaintext into ciphertext, but it would be very hard to decrypt ciphertext into plaintext. And, with a private key, it is easy and fast to decrypt ciphertext into plaintext.

In the knapsack algorithm by Merkle and Hellman, the superincreasing knapsack vector, say **Np** is chosen at random, with n elements. Also two integers, m and w are chosen, such that m is greater than the sum of the elements in vector **Np** and w is relatively prime to m. That is:

Np = (N 1, N 2, N3, …, N n)

$$m > \sum_{i=1}^{n} Ni$$

$$\gcd(w, m) = 1$$

Then the hard knapsack vector **Nu** can be constructed by multiplying the easy vector **Np** by w, modulo m.

$$\mathbf{Nu} = w\mathbf{Np} \bmod m$$

In general, the vector **Nu** will not be a superincreasing vector anymore, therefore it can be a vector for the hard knapsack problem. However, with the value of w and m, this hard knapsack problem can be converted into easy knapsack. Because w and m are relatively prime, there exists a unique multiplication inverse of w modulo m, so that $w^{-1}w = 1 \bmod m$. Therefore we can get the superincreasing vector **Nu** by:

$$\mathbf{Np} = w^{-1}\mathbf{Nu}$$

Now, the ingredients of Knapsack algorithm can be summary as following:

Private Key $\mathbf{Kp} = \{\mathbf{Np}, m, w^{-1}\}$

Here: **Np** is a superincreasing vector, $m > \sum_{i=1}^{n} Ni$, $\gcd(w, m) = 1$

$w^{-1}w = 1 \bmod m$

Public Key $\mathbf{Ku} = \{\mathbf{Nu}\}$

Given plaintext, i.e. a vector **x**, the encryption will be implemented by calculating the sum S:

$$S = \mathbf{Nu} \bullet \mathbf{x}$$

To decrypt the ciphertext 'S', it will do following conversion:

$$w^{-1} S = w^{-1}\mathbf{Nu} \bullet \mathbf{x}$$

$$\Rightarrow \quad w^{-1} S = \mathbf{Np} \bullet \mathbf{x}$$

Then this becomes easy knapsack, the plaintext, i.e. the vector **x**, can be calculated easily.

In practical implementations, the length of vectors should be large enough to withstand brute-force attack. Real knapsacks should contain at least 250 items. However, the knapsacks algorithm was broken soon after it was invented. There have been lots of variants of the knapsack algorithm since the original Merkle-Hellman knapsack algorithm was broken, but nearly all of them have also been broken, only a few of them still remain unbroken. The newer multistage knapsack (Hussain et al., 1991) has not yet been broken, another still secure variant of knapsack can be found in (Cai & Zhao, 1994). Although the Chor-Rivest knapsack (Chor & Rivest, 1984) is currently secure, the amount of computation required makes it far less useful than other algorithms. Considering that nearly all the other variants of knapsack fell, it does not seem prudent to trust those variants, which are still unbroken.

2.2.2 The RSA Algorithm

The RSA algorithm was invented by Ron Rivest, Adi Shamir, and Len Adleman (Rivest et al., 1978, 1979) at MIT soon after Merkle's knapsack algorithm was invented. It works for both encryption and digital signatures. Of all the public-key algorithms proposed over the years, RSA is by far the easiest to understand and implement, and it is the only widely accepted and implemented public-key algorithm. Since its invented, RSA has withstood extensive cryptanalysis and is still unbroken.

The RSA algorithm is a block cipher, the plaintext and ciphertext are integers between 0 and n - 1 for some value n. In the RSA algorithm, to generate the public key and private keys, we need to randomly choose two very large prime numbers, p and q. In order to maximize security, the length of p and q should be similar. Then the product of p and q is n:

$$n = pq$$

To understand RSA algorithm, one should have some knowledge of ***Number Theory***. An important quantity in number theory, refered to as ***Euler's totient function***, and written as $\phi(n)$, is the number of positive integers less than n and relatively prime to n. According to number theory we have:

$$\phi(n) = \phi(pq) = (p-1)(q-1)$$

Now we can randomly choose the encryption key, e, such that $gcd(e, \phi(n)) = 1$. Then the **public key** is $\boldsymbol{Ku = \{e, n\}}$. We can use $\phi(n)$ to compute the decryption key d, such that

$$ed \equiv 1 \bmod \phi(n)$$

In other words,

$$d = e^{-1} \bmod \phi(n)$$

And we also have

$$ed = k\phi(n) + 1, \qquad k \text{ is arbitrary integer}$$

Now we have the **private key**, $\boldsymbol{Kp = \{d, n\}}$.

Before we go to demonstrate the RSA encryption and decryption, we introduce another important theorem - ***Euler's theorem***, which states that for every a and n that are relatively prime, there is:

$$a^{\phi(n)} \equiv 1 \bmod n$$

A corollary to Euler's theorem that is useful in the RSA algorithm is: given two prime numbers, p and q, and integers $n = pq$ and m, with $0 < m < n$, the following relationship holds:

$$m^{\phi(n)} = m^{(p-1)(q-1)} \equiv 1 \bmod n$$

Now we can demonstrate the validity of the RSA algorithm. Given plaintext block M and ciphertext block C, we have:

Encryption:

$$C = M^e \bmod n$$

Decryption:

$$M' = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n = M^{k\phi(n)+1} \bmod n = M \bmod n$$

The validity of RSA algorithm has been proved.

To make the RSA algorithm clearer, a short example will probably go a long way. Suppose p = 1229, q = 1597, then n = p × q = 1962713, $\phi(n) = (p-1)(q-1)$ = 1959888. Then choose e (at random) to be 553771. In that case $d = 553771^{-1} \bmod 1959888$ = 240451. Now we have public key $Ku = \{e,n\} = \{553771, 1962713\}$, and private key $Kp = \{d,n\} = \{240451, 1962713\}$. Then we can encrypt a message said M = 1387219 using public key Ku, because $m < n$, we do not need to break it into smaller blocks.

Encryption:

Ciphertext $C = M^e \bmod n$

$= 1387219^{553771} \bmod 1962713$

$= 1344047$

Decryption:

Plaintext $M = C^d \bmod n$

$= 1344047^{240451} \bmod 1962713$

$= 1387219$

There is a very good RSA algorithm simulator on the Internet, interesting reader may go to web site http://my.netian.com/~dubs37/eng_cyber_rsa.htm and try out RSA algorithm.

There are four possible approaches to cryptanalysis of the RSA algorithm:

- Brute force attack: try all possible private keys.
- Factor n into its two factor p and q, such that $n=p\times q$. This enables calculation of $\phi(n) = (p-1) \times (q-1)$, which in turn enables determination of $d=e^{-1} \bmod \phi(n)$.

- Determine $\phi(n)$ directly before determining p and q. Then the value of d can be determined by $d=e^{-1} \bmod \phi(n)$.
- Determine d directly without determining $\phi(n)$.

The approach to defense against brute-force attack is the same for RSA as for other cryptosystems, that is, to use a large key space. So the larger the number of bits in e and d, the more secure. But this will slow down the speed of key generation, encryption and decryption, because the calculations are very complex and time consuming.

Currently the strength of RSA algorithm depends on the difficulty of factoring very large number. Most discussion on cryptanalysis of RSA algorithm have also focused on factoring n into its two prime number p and q, because it is considered that determining $\phi(n)$ given n or determining d given e and n appear to be at least as time-consuming as the factoring problem. Theoretically, it is very hard to factor a very large number, the best of known algorithms (Stallings, 1995a) factors an integer n in a time proportional to:

$$L(n) = e^{\sqrt{\ln n \times \ln(\ln n)}}$$

Unless there are some breakthrough in factoring algorithms, to factor a 150-digit number will need about 10^{20} operations, thus it seems very hard to factor it. For centuries the problem of factoring has been of interest of mathematicians, and it is likely to continue to be a matter of continuing interest. There have been some efficient methods found for factoring very small classes of numbers with special properties, but the general problem of factoring large numbers is actually hard from a computational standpoint. There may come a time when we need to discard RSA in favor of some new algorithm of factoring.

Although it seems very hard to factor a large number, but it is not as hard as it used to be. In 1977, the three inventors of RSA algorithm offered a $100 reward for factoring a 129-digit number called RSA-129 (Lenstra, 2001). They predicted that might not occur for some 40 quadrillion years. The number was factored in 1994 by an international team of volunteers working over the Internet and using over 1600 computers. It only took 8 months! This result does not invalidate the use of RSA, it simple means the larger

key sizes must be used. Currently a 200-300 decimal digit is considered strong enough for all applications.

RSA Security Inc. publishes a list of factoring challenges, with cash rewards for people who are the first to factor the number. For factoring the RSA-130 number, a 130 decimal digit number published by RSA Security Inc., the reward is $11527.00 (Lenstra, 2001). Interesting reader can try to factor this RSA-130 number and collect the money.

Chapter 3: Key Management and Electronic Copyright Protection

This chapter overviews the issues concerned with key management, as well as copyright protection of electronic document. Key management is an import issue, as the cryptography algorithm is usually known by the cryptanalysis or opponent, but the keys should be kept secret by good key management, otherwise the whole system will no longer be secure. Copyright protection is becoming increasing important because of the fact that there are more and more documents in electronic format and over the Internet, and these electronic documents require to be protected from illegal copy and use.

3.1 Key Management

In practice, key management is the hardest part of computer security. To design secure cryptographic algorithms and protocols is not easy, but this can be done by a large body of academic research, there are many strong algorithm out there, you can choose a relatively good algorithm for use. But keeping the keys secret is much harder. If the cryptographic algorithm used is very strong, it is nearly impractical to attack on the algorithm. In cryptanalysis, cryptanalysts often attack on the cryptosystems by their key management system. Therefore it is necessary to discuss the issue of key management.

Key management covers many aspects – *key generation*, *key distribution*, *key verification*, *key updating*, *key storage*, *key backup* and *key destruction*, etc.

key generation

For key generation, if the chosen key is poor, then cryptanalysts may be able to find out the key easily. Some conventional encryption algorithms, like IDEA, have a weak key. Some a key is easy to guess. A good key is a key that is neither a weak key, nor a key that is easy to guess. So It is important to choose a good key when we generate a key. A random generated key is better than a manual chosen key.

Key distribution

Key distribution is the most important part of key management, it will be discussed in more detail later.

Key verification

How does a key receiver know the key is from the specific person? Suppose a person named Bob receive a key, how does he know it came from a girl named Alice and not from someone else pretending to be Alice? If a public key is received, it must be verified that this key is really from its claimed owner.

In practice, key certification serves the goal of key verification. The key verification is related to key distribution, when a public key is distributed, usually it will not only contain the public key, but contain certificates of this public key as well. The certificates will be issued by a trusted certification authority, or CA. This is a centralized key verification. Another approach is distributed key management. Distributed key management is used in PGP system. In distributed key management, the public key is signed by the friends of the key owner. The person who signed the public key are called the ***introducers***. If a public key is received, the receiver can decide whether to trust the key or not. If he/she trusts the introducers, he/she may decide to trust the validity of the key. An introducers must be sure that the key really belongs to its claim owner, maybe the introducer will make a phone call to verify it, or even require that the key be given face-to-face. The benefit of this mechanism is that there is no CA that everyone has to trust. The down side is that when receiving a public key, he/she has no guarantee that he/she know any of the introducers and therefore no guarantee that she will trust the validity of the key.

Key update

To update key regularly is needed to maximize the security of the key. No encryption key should be used for an indefinite period. It should expire automatically like passports and licenses. There are several reasons for this:

- The longer the key is used, the greater the chance that its security will be compromised.

- The longer the key is used, the greater the loss if the key is compromised. Suppose the key is used to encrypt files on the server, then the longer the key is used, more files will be encrypted used this key. Then if the key is not secure anymore, that means many of the files on server may be insecure anymore.
- The longer the key is used, the greater the temptation for someone to spend the effort necessary to break it.
- With more ciphertext encrypted with the same key, it is easier to do cryptanalysis.

There must be a policy to determines the lifetime of a key for data encryption. For a connection-based communication, usually the lifetime of the key is the period of a connection, like a telephone call.

Some keys are used to encrypt a temporary key or session key, these key do not have to be changed frequently. They are only used for key exchange, and usually generate too little ciphertext to be accessible to cryptanalysis by the cryptanalysts. However, if the security of this key is compromised, the potential loss is extreme, because lots of session keys will therefore be insecure.

Key Storage

Key should be keep in a safe place, another person should not be easy to find out, read or make copy of the key. If a user want to store a key in a computer file, then this file should be encrypted itself. However, a key will usually be a random serial of number and very long, a person's mind will nearly impossible to remember it. There is a technique to make a ROM key, that is to store the key in a ROM chip and embed the ROM chip in a piece of plastic card like smart card, then this ROM key should also be keep in a physical safe place. If the key is generated based on a easy to remember phrase, then it will be better to just remember this phrase and regenerate the key every time when the key is required.

However, a key should never appear in plaintext format outside the encrypted device.

Key Backup

A key should be backed up in a secure place just in case it is lost. If the key is used by a staff to encrypt data and files of the company, there must be a method to backup all of the keys used by all staff inside the company. Otherwise, when some keys are missing, the company will be in trouble, they will not be able to recover the key, and the data and files encrypted by these keys. This problem can be avoided in several ways. The simplest is called ***Key Escrow***. The president of a company, for example, may require all employees to write down their keys on paper and give them to the company's security officer. Then the company's security officer will lock them in a safe place, or encrypt them all with a master key. However, there is risk here. The company's security officer should be a very reliable person, he/she should not misuse all those keys. A better solution to the key recovery problem is to use a ***secret-sharing protocol*** (Huth, 2001). The idea of secret sharing is to start with a secret, and divide it into pieces called shares. Then the shares are distributed amongst users. Different people may hold different shares, for examples, the president of the company may hold 3 out of 10 shares, the security officer may hold 2 shares, department manager may hold 1 share. This will enable the pooled shares of specific subsets of users to recover or reconstruct all of the original secret, for instance, when it is required to recover a key, the protocol say that 5 out of 10 shares of the secret is required, then the combination of the president of the company and the security officer will be sufficient to recover a key.

key destruction

Keys must be updated regularly, old keys will be replaced by new keys, old keys should be destroyed. Even though old keys may not be used anymore, they are still valuable. With them, an adversary can read old messages encrypted with those keys. Therefore, old keys must be destroyed securely. A key written on paper must be destroyed security by shredding or burning the paper. A ROM which stores a key should be smashed. If the keys are stored as files in the computer, then there must be a special method to complete erase those files, even erasing them from "swap" partition of the file system and from the system backup material such as magnetic tape.

The remaining part of this section will overview the *key distribution* protocol.

3.1.1 Distribution of Secret Key

A secret key can be distributed with a trusted Key Distribution Center (KDC). Figure 3.1 is a typical scenario of secret key distribution based on (Popek & Kline, 1979). The scenario assumes that each user shares a unique master key with the KDC. This master key should be distributed in an other more secure way, or may be given by the KDC to users face-to-face. This scenario demonstrate the distribution of a session key.

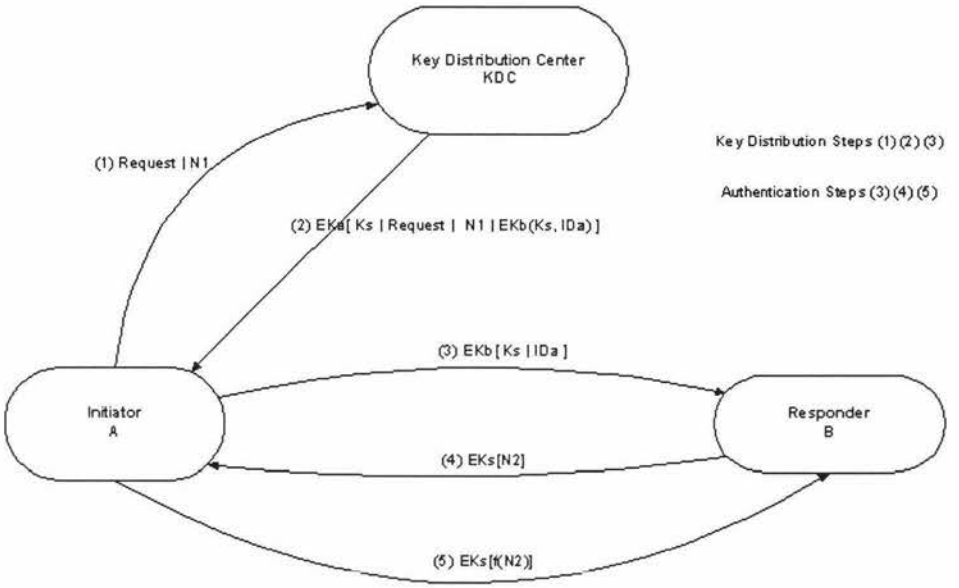


Figure 3.1: Session Key Distribution Scenario

In this scenario, A wishes to establish connection with B for data transmission. Therefore, A requires a one-time session key *Ks* to encrypt data transmitted over the network. Before the scenario, A shares a master key *Ka* with KDC , and B share its master key *Kb* with KDC. In order to request the session key from KDC and establish the connection, the following steps occur in the scenario:

1. A issues a request to the KDC. The "Request" includes identity of A and B, as well as a *nonce* N1. A *nonce* is a unique identifier for this transaction, it may be a timestamp, a counter or a random number, it should be difficult for the opponent to guess.

2. The KDC responds with a message encrypted by using *Ka*. Because only A and KDC share the master key *Ka*, thus only A can successful decrypt this message, and A is sure that the message comes from KDC. The message is:

   $E_{Ka}[\ Ks \mid Request \mid\ N1 \mid E_{Kb}(Ks, IDa)\ ]$

   It includes two parts: *Ks* | *Request* | *N1*, and $E_{Kb}(Ks, IDa)$. The former part is for A, which contains one time session key *Ks*, the original request message including the *nonce*. A will store the session key *Ks* for the upcoming data communication with B. Now, A can be sure that the request message was received by KDC correctly. This is needed because A did not encrypt the message before it sent out to KDC. The *nonce* is unique, so A can be sure this request is not a replay of other request. The other part is $E_{Kb}(Ks, IDa)$. This part contains two items, the one time session key *Ks*, and the identifier of A - *IDa*. These two items are encrypted by B's master key *Kb*. A will take this part out, and send it to B to establish the connection.
3. A take the information generated by KDC, ie. $E_{Kb}(Ks, IDa)$, and forward it to B. Because this information is encrypted with B's master key Ks, B knows the information originated at KDC. From the message, B also know that the other party is A (by Identifier IDa), and knows the one time session key *Ks*.
4. B sends a nonce *N2* encrypted by session key *Ks* to A.
5. A responds with message *f(N2)* which is also encrypted by session key *Ks*. Function *f* will performs some transformation on N2, which both A and B know. The purpose is to assure B that the reply is not the replay of the original message.

With above steps, the key can be distributed safely, and the key has also been authenticated.

3.1.2 Distribution of Public Key

Several techniques have been proposed for the distribution of public keys. They are

- Public announcement
- Public available directory
- Public-key authority
- Public-key certificates

Public announcement

Because a public key is public, the key can be sent out to any participant, or it be just broadcast to the community, or be sent to public forums like USENET newsgroup and Internet mailing list.

However, this scheme has a major weakness. Anyone can impersonate another person. For example, some user can pretend to be user A and announce the public key to the participant. The other person does not know that it is a forged one, and may encrypt data intended for A and transfer it to A. Then the forger is able to read all these encrypted message intend for A.

Public available directory

A better approach is to use a trusted public-key directory or trusted public-key database. Each public key has a registry, the integrity of which is trusted. This public-key directory will be maintain by a trusted authority. A secure channel is required for remote access to the public key registry. One method of authenticating a public key is by tree authentication of public keys.

Public-key authority

This is a tighter control over the public-key directory. The request of a public key should follow the specific protocol to ensure its security. Figure 3.2 is a public-key distribution scenario, which is based on (Popek & Kline, 1979). This scenario assumes that each participant reliable knows the public key for the public-key authority, and only the authority knows the corresponding private key. When a user, said user A, issues a request for public key from the public-key authority, the following steps occur:

1. User A sends a request message, together with a timestamp, to the public-key authority, the message containing a request for the current public key of User B.
2. The public-key authority responds with the message: $E_{KRauth}[KUb|Request|Time1]$. The message is encrypted using private key of authority, therefore user A is sure that the message originated from the authority. After decrypt this message using public key of authority, user A gets public key

of user B, i.e. *KUb*. User A also get the original request message and timestamp, so user A is assured that this message can be trusted.

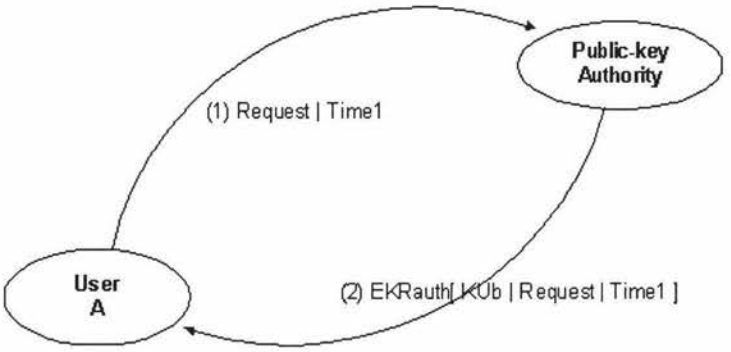


Figure 3.2: Public Key Distribution Scenario

Now user A has public key of user B, then he/she can initiate a data transmission request to user B by sent out a message like $E_{KUb}[IDa|N1]$. If user B does not know the public key of user A, he/she can request the public key for user A from public-key authority by using the same method. After user B knows the public key of user A, he/she can responses with message $E_{KUa}[N1|N2]$, after user A response to this with a message $E_{KUb}[N2]$, they are assured that the channel is secured. Now that both user A and user B know public key of each other, they can start to exchange data using public key of other party.

Public-key certificates

The scheme with public-key authority has some drawback. A user have to send requests to the authority for a public key for every other user that he/she wishes to contact, therefore, the authority may be something of a bottleneck. A better solution is to use the concept of a ***public-key certificate***. A public-key certificate is a digital certificate issued by a trusted third party called the ***certification authority*** (CA), it consists of a data part and a signature part. The data part contains the public key, identity of the key owner, and relevant information like a timestamp. The signature part contains the digital signature of the CA. The digital signature will be discussed in later in this section. To make it simple, the CA can issue the public-key certificate by only encrypting the data part using private key of CA, because the CA is the only party who knows this secret

key, the other party is assured that the certificate originates from the CA. Figure 3.3 is an approach based on (Kohnfelder, 1978) to create public-key certificates, and also for participants with the key certificates to exchange keys without contacting a public-key authority.

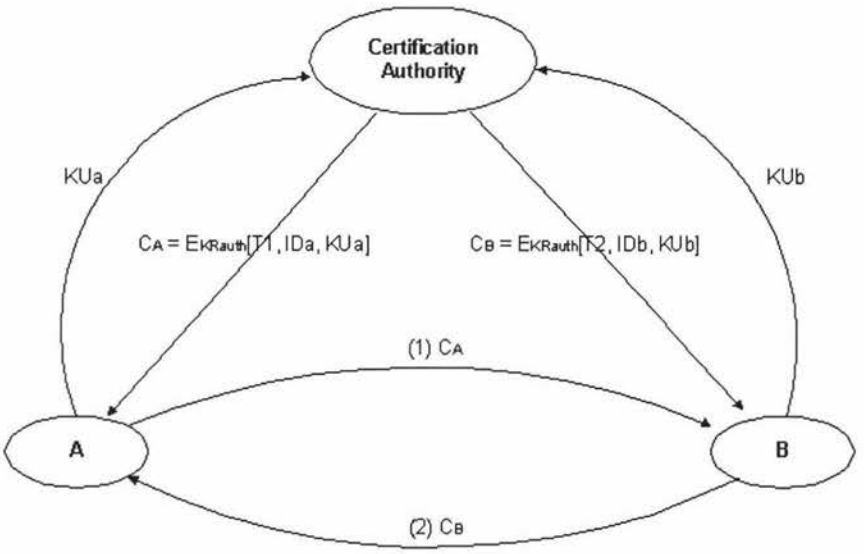


Figure 3.3: Exchange of Public-key Certificates

In this approach, any user can register his/her public key to the certification authority (CA). After CA attaches its signature to the public key, the public-key certificate is created. For user A here, his/her public-key certificate is $C_A = E_{KRauth}[T1, IDa, KUa]$. User A then can sent his/her public-key certificate to other user said user B, then user B can read the verify the certificate by decrypting the certificate using CA's public key, as follows:

$$D_{KUauth}[C_A] = D_{KUauth}[T1, IDa, KUa] = (T1, IDa, KUa)$$

The timestamp *T1* is used to validate the current of the certificate.

One format of a public-key certificate has become universally accepted: the ITU-T recommendation X.509 standard (Boeyen at al., 1999; Browne, 1997; Sun Microsystems Java Software Division, 1998). X.509 defines a framework to be used in provision of authentication services. It is based on the use of public-key cryptography and digital signatures. It does not indicate the use of a specific algorithm but recommends RSA. Figure 3.4 is the format of X.509 certificate.

Version
Serial number
Signature algorithm identifier - *Algorithm* - *Parameters*
Issuer name
Period of validity - *No before* - *No after*
Subject name
Subject's public-key information - *Algorithms* - *Parameters* - *Key*
Issuer unique identifier
Subject unique identifier
Extensions
Signature - *Algorithms* - *Parameters* - *Encrypted*

Figure 3.4: X.509 Certificate

As demonstrated in Figure 3.4, an X.509 certificate contains the following elements:

- *Version*: Identify version number – version 1, 2, and 3. It only affects what information can be specified in it.
- *Serial Number*: An serial number to distinguish it from other certificate, it should be unique within the issuing CA.
- *Signature algorithm identifier*: The algorithm used to sign the certificate by CA, together with any associated parameters.
- *Issuer name*: X.500 name of the CA entity that create and sign this certificate.
- *Period of validity*: The period that this certificate is valid, it is described by the first date and time, and last date and time.
- *Subject name*: The X.500 name of the owner of the public key who registers this public-key certificate. It should be unique across the Internet.
- *Subject's public-key information*: Contains ***public key***, the identifier of algorithm for which this key is to be used, and any associated parameters.
- *Issuer unique identifier (version 2 and 3 only)*: Unique identifier of the issuing CA. This is an optional bit string field.

- *Subject unique identifier (version 2 and 3 only)*: Unique identifier of the subject. This is an optional bit string field.
- *Extensions (version 3 only)*: A set of one or more extension fields. This field is added in version 3, it will be used to include information like: Subject alternative name, Subject directory attributes, etc.
- ***Signature***: It contains the hash code of all the other field, encrypted with the CA's private key (signature). It also includes the signature algorithm identifier.

The X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME. Therefore X.509 is an important standard.

3.1.3 Public-Key Distribution of Secret Keys

The scenario on secret key distribution depicted in Figure 3.1 has a drawback. A user has to send a request to the KDC for the session key every time when a session key is required, therefore, the KDC may be somewhat of a bottleneck.

The better solution to this is to use public key certificates based on (Needham & Schroeder, 1978). Suppose user A wants to exchange e-mail securely with anyone using conventional encryption. User A must share a unique secret key with his/her correspondent, said user B. Then A have to give the one time session key to user B. User A will do this as following:

1. User A obtained B's public key *KUb* by means of B's public-key certificate. A is assured that it is a valid key.
2. User A uses B's public key to encrypt a message to B. The message consists of an identifier of A (ID_A) and a nonce (*N1*).
3. Now, only B has the private key to decrypt message from A. So he/she can decrypt the message and get A's identifier ID_A and the nonce *N1*. B will response to this by sending a message to A encrypted with A's public key *KUa*. The message consists of *N1* and a new generated nonce (*N2*). The nonce *N1* is to assure A that its correspondent is B.

4. Now, A receive response from B, decrypt it using his/her private key and get *N1* and *N2*, then he/she is assured that the message has not been changed, and the message is from B. Then A return *N2* encrypted using B's public key, to assure B that its correspondent is A.
5. A selects a one time session key *Ks*, encrypted it using A's private key *KRa*, then encrypted it using B's public key *KUb*, then sends $M = E_{KUb}[E_{KRa}[Ks]]$ out to B.
6. B use his/her private key *KRb* and A's public key *RUa* to recover the secret key *Ks*:

   $$Ks = D_{KUa}[D_{KRb}[M]]$$

7. User A start to exchange encrypted e-mail with user B, by using the one time session key *Ks*.

This scheme ensures both confidentiality and authentication in the exchange of the one time session key.

3.2 Electronic Copyright Protection

Nowadays, more and more of the electronic multimedia documents are in digital format. To make a copy of these digital documents will be easier, the cost is cheaper, and the quality of the copy can be identical as the original document. All these have increased the potential for misuse and theft, and have significantly intensified the problems associated with copyright protection and enforcing these rights. Because the widespread use of Internet across the world, the multimedia documents providers and distributors over the Web are general afraid of online services, they are looking for technical solutions to address the challenge of copyright protection.

One approach to address the problems associated with copyright protection is *Usage Control*. This approach will restrict the control of each usage of the protected material, it is a pay by usage method. Although it is the predominant technology for specific application like pay-TV, video-on-demand etc., and it is not suitable for other application or situation because of its restrictive nature.

Another approach is the *digital copyright labeling techniques*. This technique does not restrict and control usage of protected material, it does allow unlimited copying and using, but it provides evidence in the case of copyright infringement. This approach achieves this by embeding a digital mark into the material being protected. The mark is a piece of copyright information such as owner, recipient, origin, content, logo of owner, and serial number etc. This makes it possible to provide evidence for copyright infringement, to trace illicit copying and dissemination.

The digital copyright labeling techniques do not provide the mechanism to actually prevent illegal copying and dissemination, it only serve as a kind of deterrent to illicit copying and dissemination, and serve to provide evidence of illegal usage of the protected material to the court. However, this also requires a legal system that allows the copyright holders to sue illegal acts.

There are two types of digital copyright labeling:

- *Ownership labeling*: The document is embedded with a mark that uniquely identifies the copyright holder. This allows proof of ownership of the documents. The mark is the same for all buyers of the documents. The mark is often referred to as *watermark*, and the technique of Ownership labeling as *watermarking*.
- *Recipient labeling*: Each copy of the document sold or distributed is embedded with a mark, like a serial number, to identify the buyers. This allows its distribution to be uniquely traced. The mark will depends on the buyer's identity. This mark is often referred to as *fingerprint*, and the technique of Recipient labeling as *fingerprinting*.

3.2.1 **Watermarking** (Lacy et al., 1998; Oppliger, 1999)

The term *watermarking* comes from a technique in physical world that a specific image or text is impressed into papers during the process of making paper in the manufactory. The purpose of watermarking is to guarantee the authentication, quality and ownership.

In information technology world, electronic watermark is a mark (a piece of the message) added to digital document, such as image document, audio file, etc. The

watermark can be extract later from the digital document. The purpose of this is to guarantee the integrity of the document, to assertion the ownership, to be used for usage control, etc.

Watermark insertion

As illustrated in Figure 3.5, the inputs to the watermark insertion are a user key *Ks*, a watermark *Es*, and the object *I* in which the watermark to be inserted. The output is a watermarked object *I′*. The procedure to insert the watermark depends on the watermarking technique in use.

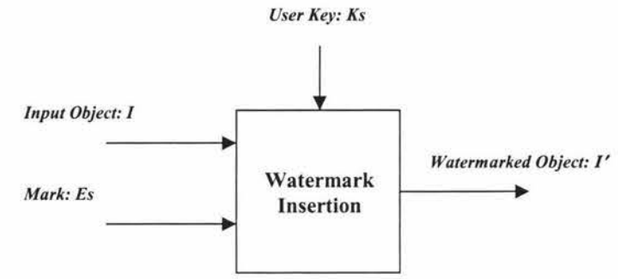


Figure 3.5: Watermark Insertion

The watermark embedded in the electronic document should be recoverable. The watermark can either be detected or extracted, depending on the nature of the watermark and the technique used to embedded the watermark.

Watermark Extraction

In some watermarking scheme, a watermark can be extracted in its exact form. As illustrated in Figure 3.6, the watermark extraction takes input of user key *Ks* and a watermarked object *I′*, then it will extract the watermark embedded inside the object.

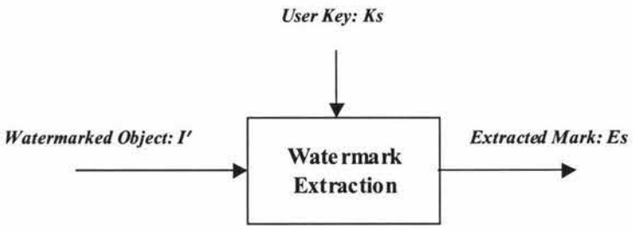


Figure 3.6: Watermark Extraction

Watermark detection

In other watermarking schemes, a watermarking can only be detected whether a specific watermarking signal is present in an electronic document. As illustrated in Figure 3.7, the watermark detection takes input of user key *Ks*, a watermarked object *I′*, and Watermarked related information – maybe a watermark identify, and produces as output a binary decision, either "yes" or "not", depending on whether a specific watermark is included in the document or not.

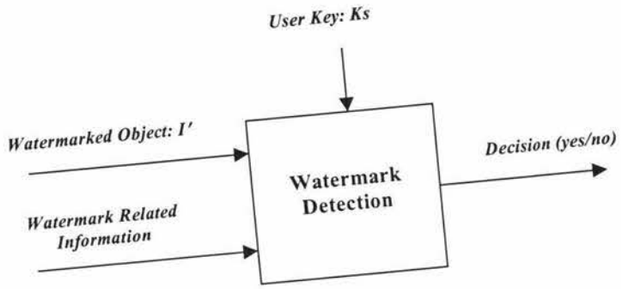


Figure 3.7: Watermark Detection

A watermark may be either a *visible watermark*, or an *invisible watermark*. An invisible watermark is imperceptible but can be detected by appropriate software. Usually users prefer invisible watermark.

Watermarking techniques are classified as being either *fragile* or *robust*. A fragile watermark will be corrupted after the document being processed. A robust watermark can resist transformation. Robustness is a key requirement for the watermarking techniques. But it is hard to design a robust technique that can satisfy all the requirement of watermarking.

Watermarking can be either a *public watermark* or a *private watermark*. A public watermark can be detected and read by anyone using appropriate software. A private watermark can only be detected and read by the person who own some secret information related to the watermark. From the security point of view, a private watermark is better than a public watermark. But in a private watermark, the secret information should be delivered to the user or third party via secure channel. Actually private watermarking is more suitable used to demonstrate the ownership of the

Watermark detection

In other watermarking schemes, a watermarking can only be detected whether a specific watermarking signal is present in an electronic document. As illustrated in Figure 3.7, the watermark detection takes input of user key *Ks*, a watermarked object *I'*, and Watermarked related information – maybe a watermark identify, and produces as output a binary decision, either "yes" or "not", depending on whether a specific watermark is included in the document or not.

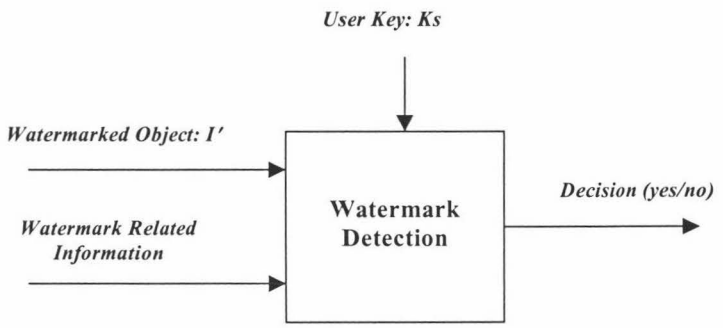


Figure 3.7: Watermark Detection

A watermark may be either a *visible watermark*, or an *invisible watermark*. An invisible watermark is imperceptible but can be detected by appropriate software. Usually users prefer invisible watermark.

Watermarking techniques are classified as being either *fragile* or *robust*. A fragile watermark will be corrupted after the document being processed. A robust watermark can resist transformation. Robustness is a key requirement for the watermarking techniques. But it is hard to design a robust technique that can satisfy all the requirement of watermarking.

Watermarking can be either a *public watermark* or a *private watermark*. A public watermark can be detected and read by anyone using appropriate software. A private watermark can only be detected and read by the person who own some secret information related to the watermark. From the security point of view, a private watermark is better than a public watermark. But in a private watermark, the secret information should be delivered to the user or third party via secure channel. Actually private watermarking is more suitable used to demonstrate the ownership of the

documents. A public watermark is more suitable used to detect copyright violations in such documents as images published on the Web.

The private watermarking can be classified as being *secret key watermarking* or *public key watermarking*. A secret key watermarking uses the same key for watermarking insertion and watermarking extraction or detection. The key is shared by the document owner and the document receiver. A public key watermarking uses a pair of public-key/private-key, the private key is used in watermarking insertion, and public key is used in watermarking extraction or detection. In this scheme, the owner of the document embeds the watermark into the documents, all other persons know the public key so they are able to extraction or detection the watermark.

The techniques to satisfying the requirement imposed on robust watermarking is the subject of current research and development, some of the articles on this subject can be found in (Fridrich, 1998; Herrige et al., 1998; Watanabe & Kasami, 1997).

3.2.2 **Fingerprinting** (Joancomarti, 2001)

Similar to the way in the physical world that fingerprints make people unique, fingerprinting a electronic document will make the document unique, i.e., each copy of the document sold will be embedded with a individual mark that make this copy unique. Therefore, if an illegal copy turns up, the document provider can detect the fingerprint and trace it to find out the original copy that was redistributed.

The techniques used to embed a fingerprint into a document are the same as to embed a watermark, so we do not repeat them here.

Fingerprint techniques can be classified as:

- *Classical fingerprinting* (Boneh & Shaw, 1995): This is a symmetric scheme, both the document provider and buyer know the fingerprint of the copy. The weakness of this scheme is that even if document provider identifies a dishonest buyer from redistributed copy, the previous knowledge of document provider prevents him/her to use them as a proof in the court.

- *Asymmetric fingerprinting* (Pfitzmann & Schunter, 1996): Only the buyer knows the content of fingerprint, the document provider produces and embeds the fingerprint into document without knowing the content of fingerprint. However, even if a dishonest buyer is identified, the document provider still can not use it as proof, because the document provider know the identity, he/she may be able to reproduce the fingerprint.
- *Anonymous fingerprinting* (Pfitzmann & Waidner, 1997): The document provider does not know the fingerprint or the identity, each fingerprint copy is provided by a trusted third party as a *registration authority*. Upon identifying a redistributed copy, the document provider needs help from registration authority to identify the origin of the redistributed copy. In this case, the fingerprint can be used as proof in the court.
- *Collusion-Secure fingerprinting* (Boneh & Shaw, 1995): This scheme guarantee that the fingerprint can not be destroyed even if up to c buyers collude. Here c is the security parameter.

Chapter 4: Design Decision of Security System for AudioGraph

This chapter will first demonstrate two examples that use cryptography algorithm to achieve security. One example is a password protection system used in UNIX system, the other one is a Security Socket Layer (SSL) system using over the Internet especially over the World Wide Web. Then the proposed security system for our AudioGraph teaching tools will be demonstrated, following by discussion of the security system as well as the selection of cryptographic algorithms. The last part of this chapter will analyze the degree of security of our system against brute-force attacks and cryptographic algorithms attacks.

4.1 Examples of Security System

In today's digital world, information security can be classified as:

- *Data (file) Security*: This is the security consideration of data and files stored in in storage device like floppy disk, hard disk or CD-ROM etc. To make the data or files secure, one must hide the contents of the data or files by using cryptographic algorithm to encrypt the contents of data or files.
- *System Security*: This is the security consideration of the operating system. It is the responsibility of the operating system designer and the system administrator. The system designers must consider the security of the system, must provide access control, like password protection, and other security tools to the system. System administrators are required to detect intruders and virus and make action on them, must establish security policy of the system, must build a firewall to protect the system from intruders, etc.
- *Network Security*: This is the security consideration of data and file transmission over the network. Network security covers *Authentication*, *Electronic Mail Security*, *IP Security*, *Web Security* and *Network Management Security*.

The requirement of AudioGraph security system must provide password protection for usage control. Because AudioGraph presentation material can be played back by Web

browsers over the Internet, so we must also consider its network security. Therefore, this section give two examples, one is password protection system of UNIX operating system, another example is the most popular security standard in the Internet: the Secure Socket Layer (SSL).

4.1.1 Password Protection in UNIX System

In UNIX operating system, a user must provide not only a user name but also a password on order to get into the system. Actually all multi-user systems do it the same. The username serves as the unique identity of an individual user who is logging on to the system. The password serves to authenticate the identity.

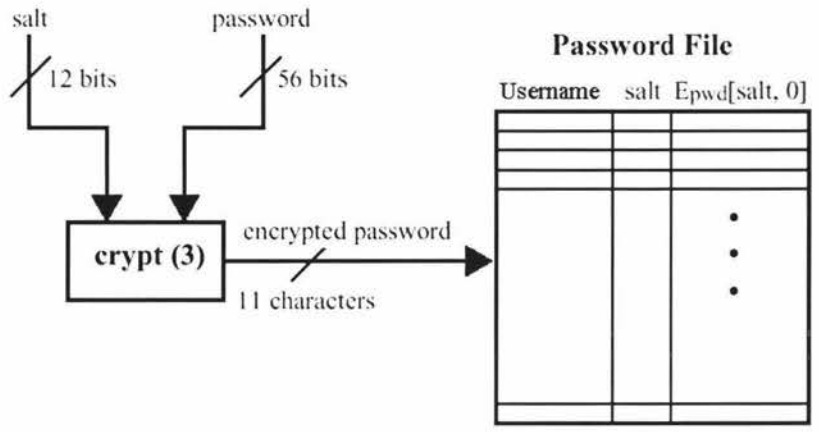
The password protection system works as an access control system to control the user's logging on. When the UNIX system gets your user name and password, it will do verification of its validity. If the user name and password are both valid, it will let the user log on, otherwise it will forbid the user from logging on. The system may then give different users different privileges. The password system should be secure not only from the non-authorized user penetrating or gaining access to the the system, but also against authorized user to access to data, program or resources for which such access is not authorized.

UNIX uses the file **/etc/passwd** to keep track of every user on the system. The file contains the username, real name, identification information, and basic account information for each user. Each line in the file contains a record of one user, the record fields are separated by a colon, following are some examples of the file lines:

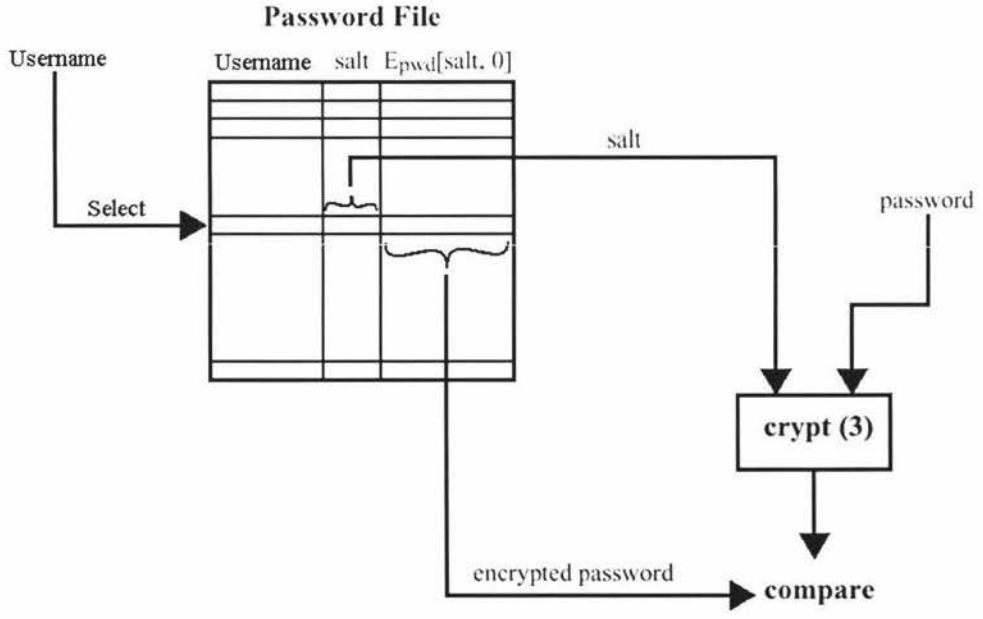
```
Root:fi3sED95ibqR6:0:1:System Operator:/:/bin/csh
Daemon:*:1:1::/tmp:
UUCP:OORoMN9FyZfNE:4:4::/usr/spool/uucppublic:/usr/lib/uucp/uucico
Rachel:eH5/.mj7NB3dx:181:100:Rachel Cohen:/u/rachel:/bin/csh
```

The first field is the username, the second field is the password of the username. Here you may notice that the password is not in cleartext, it is in ciphertext format with 13 character, the first two character of this field is the *Salt*, the remaining 11 character is the

user's encrypted password. The algorithm to convert a password into ciphertext format is ***crypt(3)***, a variant of DES.



(a) Loading a new password



(b) Verifying a password

Figure 4.1: UNIX Password Protection System

Figure 4.1 demonstrates UNIX password protection system. In this system, each user selects a password of up to 8 characters in length, and then this is converted into a 56-bits value (each character has 7-bits length, together they are 56 bits length). Then this

56-bits value is used as a key to an encryption routine. The system also generates a 12-bits value called *salt* to input to the encryption routine. The value of *salt* relates to the time the password was assigned and serves to make brute-force key attack more difficult. The encryption routine, known as ***crypt(3)***, is based on DES. In crypt(3), it takes the 12-bits *salt* value to modify the E-table in DES algorithm (more specifically if bit i of *salt* is 1, then bit i and bit i+24 are swap in the E-table), then it takes the 56-bits value from user's password as a key to encrypt a 64-bits block of zeros. The resulting 64-bits block of cipher text is then encrypted again with the same key. This process is repeated a total of 25 times. The final 64 bits are unpacked into a string of 11 printable characters that are stored in the password file (i.e., file **/etc/passwd**). The 12-bits *salt* is converted to 2 characters and stored in the password file as well (the first two character in the password filed).

When a user attempts to log on to a UNIX system, the user input a username and a password. The system uses the username to index into the password file **/etc/passwd** and retrieve the encrypted password as well as the plaintext *salt*. Then system uses the salt and the password from user as input to crypt(3) encryption routine, if the output of crypt(3) matches the encrypted password from file **/etc/passwd**, the password is accepted and system just let user logging on. Refer to Figure 4.1(b).

The UNIX password protection system is a good example, it show us how to use a cryptography algorithm in a security system. This section only overviews the password protection of the UNIX system, if you are interesting in UNIX security, more detail can be found in (Garfinkel & Spafford, 1991).

4.1.2 Security Sockets Layer (SSL) and Transport Layer Security (TLS)

Secure Sockets Layer (or SSL) is a flexible, general-purpose encryption system. It was first introduced in 1994 with the first version of the Netscape Navigator browser. Since then, SSL has been widely accepted and now it became the predominant secure protocol on the Web. The latest version of SSL is SSL v3.0 proposed in 1996. Nowadays most of the commercial Web servers and Web browser support SSL3.0. For details of SSL3.0, refer to (Freier et al., 1996).

In 1996, The Internet Engineering Task Force (IETF) chartered a Transport Layer Security Working Group (TSL WG) within the Security and Transport Areas. The first document released as an Internet Draft is TLS 1.0. The TLS 1.0 protocol is based on the SSL3.0, it made some modification to the SSL v3.0. For details of TLS1.0, refer to (Dierks & Allen, 1999).

In the TCP/IP protocol architecture, SSL/TLS is layered on top of the TCP layer, as depicted below:

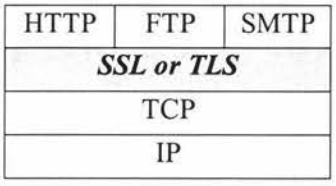


Figure 4.2: SSL/TLS in the TCP/IP protocol architecture

The SSL/TLS layer provides security for the application layer. In most of the implementations, SSL/TLS is embedded into the application software, for example, SSL is implemented in Netscape Navigator and Internet Explorer.

SSL/TLS is designed to provide secure end-to-end service, it has two layers of protocols, as illustrated in Figure 4.3. The lower layer, *SSL record protocol*, receives data from higher layer SSL subprotocols and addresses data fragmentation, compression, authentication, and encryption. The other SSL subprotocols are the payloads of the SSL record protocol.

The *SSL handshake protocol* is the main SSL subprotocol that is layered on top of the SSL record protocol. The purpose of this protocol is to let the client and server agree on a common SSL protocol version and to select the compression method and cipher specification, to authenticate each other. It creates a pre-master secret from which the master secret and various session keys for message authentication and encryption will be derived. The procedure of handshaking will be demonstrated later.

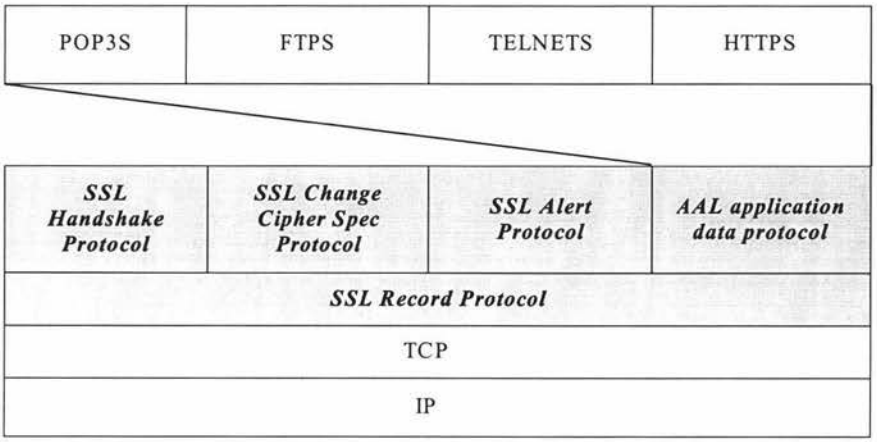


Figure 4.3: SSL/TLS Protocol Architecture

SSL Change Cipher Spec Protocol consists of a single byte with value 1, its purpose is to cause the change of state. *SSL Alert Protocol* is used to convey SSL-related alerts to the peer entity. *AAL application data protocol* is used to encapsulate data from higher layer, such as HTTPS.

Table 4.1: Reserved Port Numbers Assigned for Application Protocols That Run on Top of TLS/SSL (Internet Assigned Numbers Authority, 2001)

Keyword	*Port*	*Description*
nsiiops	261	IIOP name service over TLS/SSL
https	443	HTTP over TLS/SSL
smtps	465	SMTP over TLS/SSL
nntps	563	NNTP over TLS/SSL
ldaps	636	LDAP over TLS/SSL
ftps-data	989	FTP (data) over TLS/SSL
ftps	990	FTP (control) over TLS/SSL
telnets	992	TELNET over TLS/SSL
imaps	993	IMAP4 over TLS/SSL
ircs	994	IRC over TLS/SSL
pop3s	995	POP3 over TLS/SSL

There are several possibilities for an application protocol to make use of SSL protection. In practice, separate port numbers have been reserved and assigned by Internet Assigned Numbers Authority (IANA) for application protocols that may run on top of TLS/SSL. In this way, an application protocol may be assigned two reserved port number, one is for application protocol running on TLS/SSL. For example, the port number for HTTP

over TLS/SSL is 443, another one is a normal port number which does not run on TSL/SSL, e.g., the port number for normal HTTP protocol is 80 (Internet Assigned Numbers Authority, 2001). Table 4.1 summarizes the reserved port numbers assigned by IANA for the most popular application protocols.

SSL Handshake Protocol

As mention earlier in this section, the SSL handshake protocol is the main subprotocol of SSL/TLS. Actually it is also the most complex part of the SSL protocol suites. The Handshake Protocol consists of a series of message exchanged by client and server. For full technical details, consult the SSL specification (Freier et al., 1996). Briefly the steps of the process are (refer to Figure 4.4):

1. *Client sends "client_hello" message*: It contains the parameters of
   - *Version* : The highest SSL version understood by the client
   - *random* : Consists of 32-bits timestamp and 28 bytes random number, these serve as a nounces (a nounce is a number used to uniquely identify this transaction)
   - *session_id* : here it is zero, to indicate that the client wishes to establish a new connection on a new session
   - *cipher_suite* : The list of cryptographic algorithms supported by this client.
   - *compression*: The list of compression methods the client supports.

   After sending this message, the client waits for the "*server_hello"* message from the server.
2. *Server responds with* a *"server_hello" message*: This is the response from the server. The *version* field contained in this message is the lower of the version suggested by the client and the highest supported by the server; the *random* field is a nounce generated by server; the *session_id* field contains the value of new session; the *cipher_suite* field contains a single cipher suite selected by the server from those proposed by the client; the *compression* field contains the compression method selected by the server from those proposed by the client.
3. *Server sends "certificate" message*: If server is using certificated-based authentication, the server sends its signed X.509 v3 site certificate to the client.

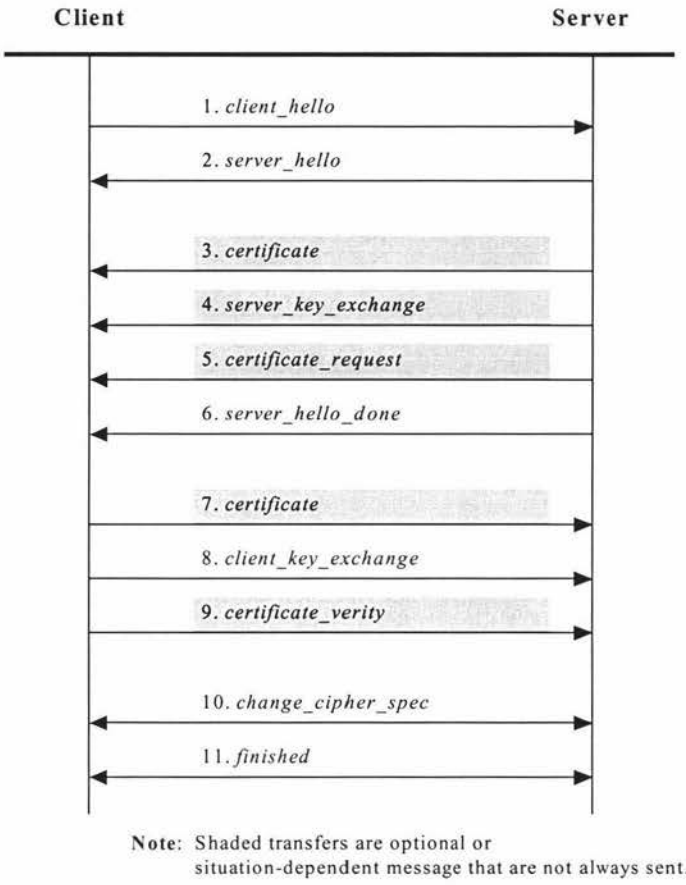


Figure 4.4: The SSL Handshake Protocol Action

4. *Server may send "server_key_exchange" message*: If it is required, the server may send this message to the client. If the server is using a certificated-based authentication, this is not required.
5. *Server may send "certificate_request" message*: If client certificates are used for client authentication (currently rare), the server next sends this message to client request the certificate.
6. *Server sends "server_hello_done" message:* This message is always sent, it is to indicate the end of the server hello and the associated message, and after this the server will wait for response from client.
7. *Client may send "certificate" message*: If the server has requested a certificate, then this is the time to send the client certificate. If no suitable certificate is available, the client sends a *no_certificate* alert instead.
8. *Client sends "client_key_exchange" message* : This is where the conventional secret key comes from. The details vary depending on the cipher suite chosen. In

the most typical case, the client generates a *pre-master secret*, and then encrypts it with the public key from the server's certificate or a temporary RSA key from a server_key_exchange message, then client sends it to server. This pre-master secret is used by both client and server to generate the master secret that is then used to generate the session key.

9. *Client may send "certificate_verity" message*: This message is only sent following any client certificate that has signing capability. The message signs a hash code based on the preceding messages.
10. *Both Client and Server send "change_cipher_spec" message*: This is a message to simple confirms that both client and server are ready to start communication using the agreed conventional encryption algorithm and session key.
11. *Both Client and Server send "finished" message*: This message verified that the key exchange and authentication processes were successful.

Using SSL

On the client site, it is easy to use. Netscape Navigator, Microsoft Internet Explorer and other popular web browsers offer support for the SSL protocol. To see this in action, point one of these browsers at the URL https://www.nzedsoft.com, the browser will establish a secure connection with the Web server of New Zealand Educational Software Center, the application protocol here is HTTPS, i.e. the HTTP running on SSL. Figure 4.5 shows how this appears in Microsoft Internet Explorer. You will notice that there is a *lock icon* at the bottom right of the browser window, which is not usually displayed. This means that the browser has established a secure connection with the Web server. To get more information about the certificate from the server, you can double click on this lock icon, then a window like Figure 4.6 will appear, you may find out that the certificate information is in X.509 v.3 format.

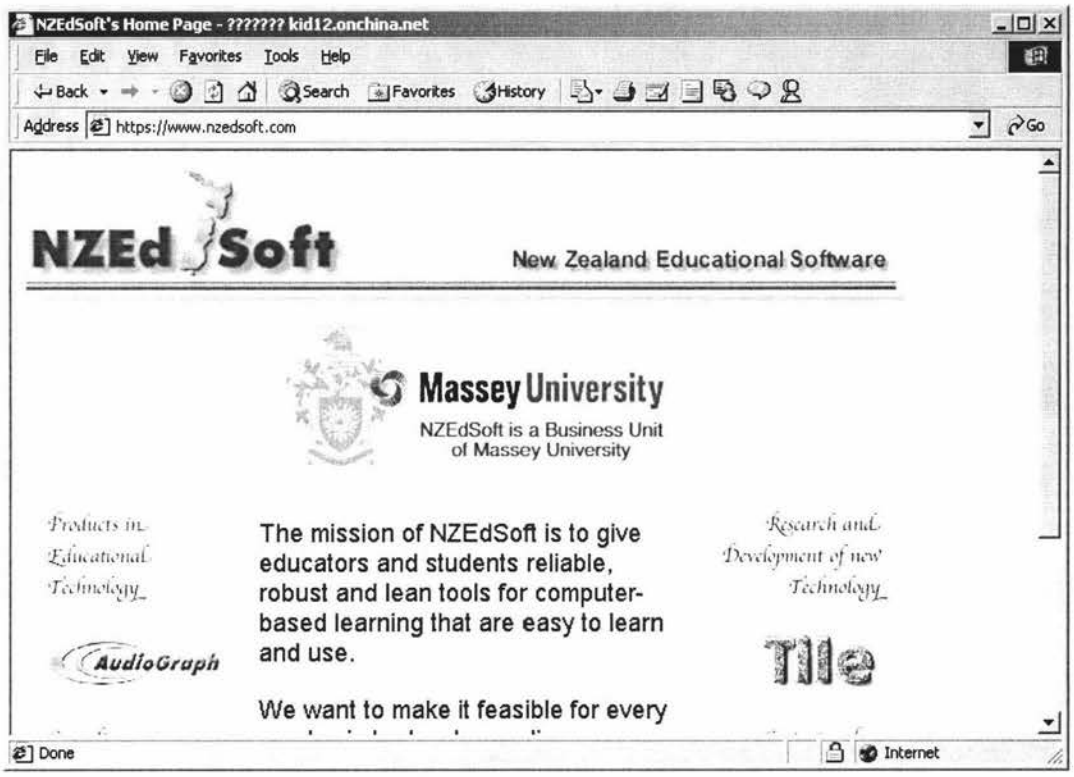


Figure 4.5: The NZEDSoft site uses SSL to keep delivery information confidential

Figure 4.6: Certificate of NZEDSoft Web site

On server site, in order to use SSL, the Web server must be a SSL-enabled server. Now most of commercial Web servers support SSL. SSL-enabled servers are available on

UNIX, Macintosh, Windows, and OS/2 operating systems. In order to make use of SSL with a Web server, a site certificate must be obtained and installed to the Web server. This thesis will not discuss this topic any further, interested readers may refer to the manual of the various Web servers. If a user want to obtain a site certificate, he/she may go to URL:

https://www.thawte.com/, or http://www.verisign.com/, or http://www.globalsign.net/ for information on how to obtain an X.509 certificate.

4.2 Proposed Security System For AudioGraph Multimedia Teaching Tools

Section 1.5.1 has already described the requirements of the security consideration for AudioGraph multimedia teaching tools. Figure 1.5 also briefly illustrated the security system. In this section a proposed implementation of the security system will be demonstrated in detail.

4.2.1 Structure of AudioGraph Security System

The security system proposed includes three parts, one is the *Copy Protection record* embedded in the .aep files. The second part is a tool called the *Key Insertion Tool,* which is used to insert the Copy Protection record into .aep files. Finally, the third part is a usage control mechanism, which is embedded into the AudioGraph Plug-in, such that when the AudioGraph Plug-in comes to the Copy Protection record of an .aep file, it will do usage control based on userID an password from the user's input.

The format and function of *Copy Protection record* inside the .aep file has been explained in section 1.5.2, we do not repeat them here. This section will demonstrate the Key Insertion Tool and the Usage Control mechanism in the AudioGraph Plug-in.

Key Insertion Tool

The current AudioGraph Recorder which is used to produce courseware materials does not embed the function to insert Copy Protection records into .aep files. This is all right

for current use where copy protection does not required. However, when copy protection is required, there must be a tool to insert *Copy Protection record*s into .aep files, this tool can be an independent application, which can be integrated into the AudioGraph Recorder if required. Figure 4.7 illustrates the functions of *Key Insertion Tool*.

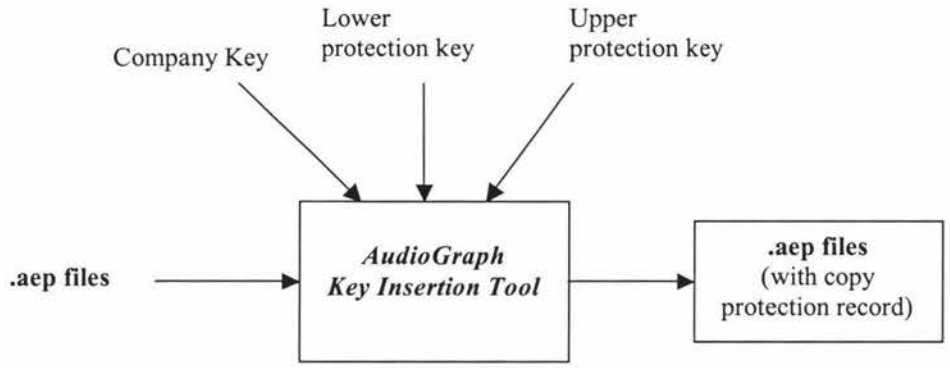


Figure 4.7: AudioGraph Key Insertion

The inputs to the Key Insertion Tool are Company Key, Lower Protection Key, Upper Protection Key, and .aep files. It will create a Copy Protection record and then insert it into .aep files. If the Copy Protection record has already been inserted into the input files, then the Key Insertion Tool will just work to change the content of the Copy Protection record according to value of Company Key, Lower Protection Key and Upper Protection Key.

The Key Insertion Tool can be invoked by the users who want to insert Copy Protection record into .aep files or make change to the record. If required, it can be integrated into AudioGraph Recorder. A AudioGraph Recorder integrated with this tool should be able to export the presentation into .aep files with a propriety Copy Protection record. (The tool can also be used to provide an on-line service to clients who wish to protect their AudioGraph material.)

Scheme of Usage Control in AudioGraph Plug-in

The Scheme of Usage Control in AudioGraph Plug-in is illustrated in Figure 4.8. On playing back the presentation over Web browser, when the AudioGraph Plug-in encounters a Copy Protection record, it will retrieve the three key from the record:

Company Key, Lower Protection Key and Upper Protection key. These three keys will be encrypted by Tiny Encryption Algorithm (TEA) algorithm using a internal TEA key. The output of TEA encryption will be used as a key to the Blowfish encryption (TEA and Blowfish algorithm will be discussed in chapter 5). Then the userID from user's input is encrypted by Blowfish encryption algorithm, the output of Blowfish encryption should be the correct password for this specific userID in our system. The system will compare the output value from Blowfish encryption to the password from the user's input, if they are match, then the password from user's input is correct, and AudioGraph Plug-in will continue to playing back the presentation, otherwise the system will prohibit the playing back of the materials by displaying a warning message and then the Web browser will exit.

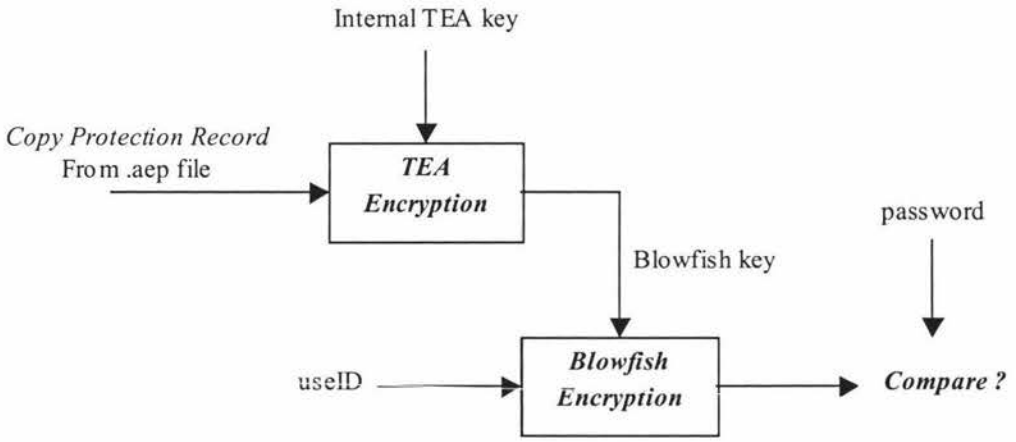


Figure 4.8: Scheme of Usage Control in AudioGraph Plug-in

Details of TEA and Blowfish cryptographic algorithms will be explained in next the chapter. The internal TEA key in Figure 4.8 is a 128-bits random number, which is coded into the system. It should be treat securely.

This scheme does not consider one practical situation, i.e., every time the user plays back a presentations, the user would have to input his/her userID and password, even when the user clicks on the "Refresh" button for Internet Explorer or "Reload" button for Netscape Navigator. This is even the case, when the user just goes to the very beginning of the presentation. Even worse, the userID and password may be just random characters for security reason. By asking the user to input userID and password again

and again, user may possibly make the user really annoyed. Therefore, this scheme needs to be improved.

An improved scheme is to store the userID and password on the local disk if user has ever entered a correct userID and password pair. Then every time when the userID and password is required, it will check from the local disk. If there is a userID and password on local Disk, it will retrieve the userID and password, then check whether this password is correct for the specific userID. If the password is correct, then it will continue to play back the presentations, otherwise it will ask user to enter userID/password again. Every time when a new pair of userID/password is entered, the old pair of userID/password on local disk will be replaced. If a user has never entered a userID and password before, then it will not find a userID and password on local disk, on this situation it will just ask user to enter userID and password.

By using this improved scheme, users will only need to enter the userID and password once for a given set of keys, then on playingback, all the presentations, which have the same three keys (Company Key, Lower Protection Key and Upper Protection Key), would not require the user to enter the userID and password again.

However this improved scheme raises a risk, which is the plaintext format of userID and password stored in local disk. Although the system usually has access control to protect the data, files and resource from being illegal access, we still need to protect this userID and password from being exposed to unauthorized user, just in case the unauthorized user penetrate into the system access control somehow. The approach to this is to encrypt the userID and password before it is been saved to a local disk file. The improved scheme can be depicted clearly using Flow Diagram, as illustrated in Figure 4.9.

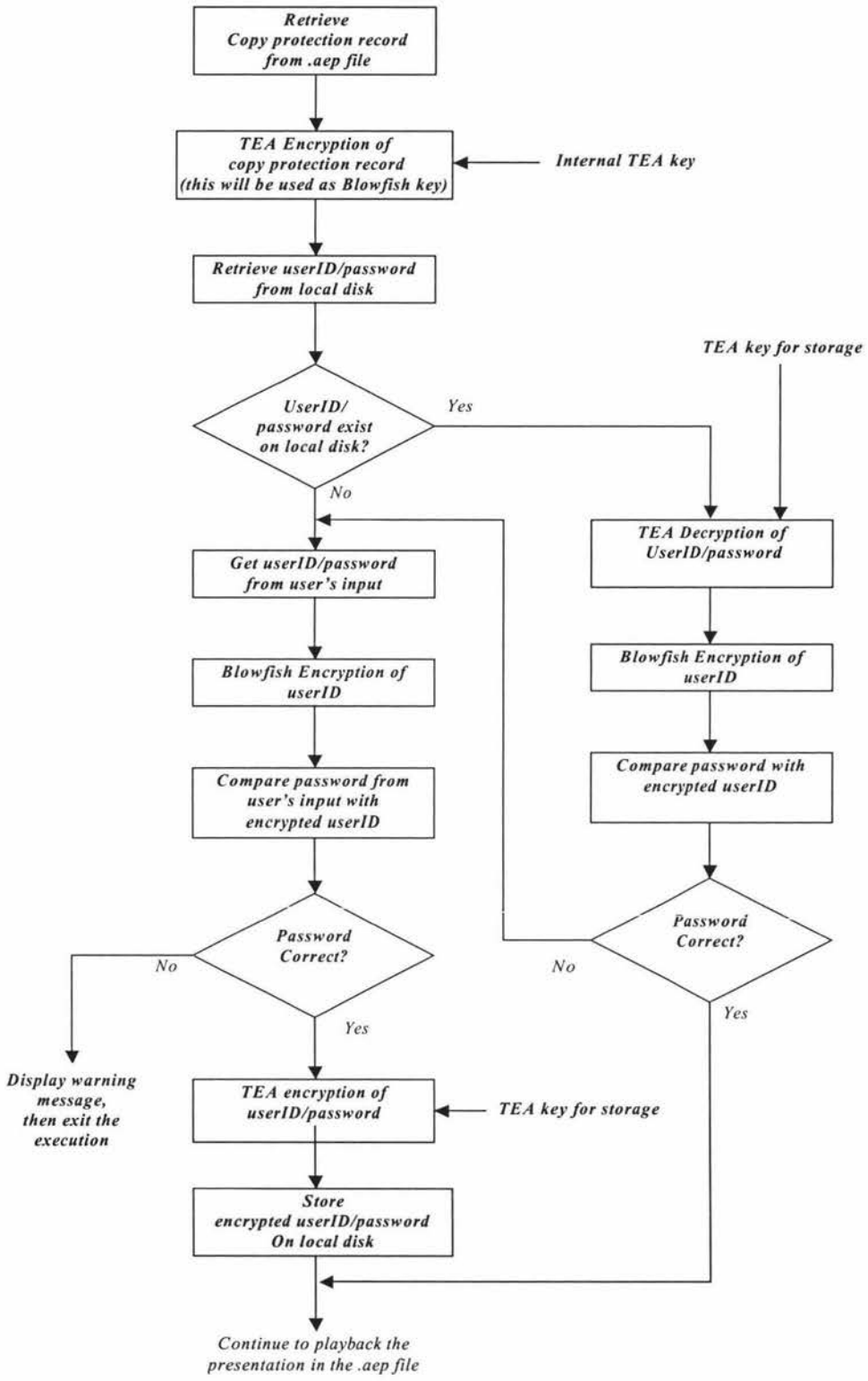


Figure 4.9: Flow Diagram of Usage Control in AudioGraph Plug-in

4.2.2 Consideration of Proposed Security System for the AudioGraph Multimedia Teaching Tool

This section will discuss the proposed security system for the AudioGraph. It will discuss the Copy Protection Record, the Key Insertion Tool, and the design decision of the Usage Control in AudioGraph Plug-in.

Copy Protection Record

In previous chapter, two digital labeling techniques were discussed: watermarking and fingerprinting. The Copy Protection record embedded in .aep files contains not only a mark as an identity of ownership (Company Key), but contains a mark as identity of recipient (Lower Protection Key and Upper Protection Key) as well, therefore it is the combination of watermarking and fingerprinting.

The essential purpose of this Copy Protection record is to protect the presentation material from being used by unauthorized persons. The side function of this Copy Protection record is to serve as owner and recipient labeling. As mentioned earlier, when playing back such materials, a userID and password are required. However, there is a possibility that both the material and the userID/password are illegally redistributed.I In this case a specific tool can be used to identify the Company Key, Lower Protection Key and Upper Protection Key, and therefore it can be used to trace the origin of the redistribution according to these three keys.

However, when an illegal copy is identified, the evidence can not be used as a proof in court, the reason is the same as for fingerprinting, that is the knowledge of the AudioGraph material producer who also knows these key and he/she is also able to redistribute the materials. Therefore a strong cooperation between the AudioGraph material producer and the material distributor is required in order to protect the material from been illegally copied and redistributed.

Key Insertion Tool

The Key Insertion Tool here is an independent application, which can be invoked by the end user or by another application. Its main function is to insert a Copy Protection record into .aep files.

To insert a Copy Protection record into a .aep file, the user has to specify an .aep file folder and a file name, as well as Company Key, Lower Protection Key and Upper Protection Key. As mention earlier, it will check the existence of the Copy Protection record first. On finding a Copy Protection record, a message to inform the user will be displayed, and it will also display the contents of Company Key, Lower Protection Key and Upper Protection Key. Therefore, *this tool can be used to detect and extract Copy Protection record* from the .aep files.

The userID and its corresponding password are related to this Copy Protection key (refer to Figure 4.8). Actually a password for the specific userID is the ciphertext of the userID, the ciphertext is calculated by the Blowfish algorithm using an encryption key derived from the Copy Protection key. Therefore, if a user wants to know the password for a specific userID, he/she also needs to know the Copy Protection record as well. Thus it is very natural to combine the function of determining the password into this Key Insertion Tool.

In conclusion, three functions should be implemented for the Key Insertion Tool:

1. Insert Copy Protection record into .aep files
2. Detect and Extract Copy Protection record from .aep files
3. Determine password for specific userID

Scheme of Usage Control in the AudioGraph Plug-in

Recall the UNIX protection system demonstrated in section 4.1.1, it uses the file **/etc/passwd** to keep track of every user on the system, every user has an entry in the file. An entry contains the username, password, and other messages related to the user. If it cannot find the entry of a given username, the UNIX system does not allow the users

with this username to log on. However, this method is not suitable for AudioGraph. The AudioGraph (refer back to section 1.5) security system does not allow us to have a database or file to keep track of every user, because those AudioGraph presentations materials are either on the Web server or on CD-ROM, which can be carried away by the users. It will therefore have to be derived from the password based on the userID and the Copy Protection record embedded in .aep files. There are two possible methods for AudioGraph security system to calculate a password given userID and Copy Protection record:

1. Using Copy Protection record as key to encrypt userID, the ciphertext of userID is used as password.
2. Using userID as a key to encrypt Copy Protection record, the ciphertext of Copy Protection record is used as password.

Either way, the Copy Protection record should be kept a secret. From a security point of view, there is no significant difference between these two methods, it is quite natural for the AudioGraph security system to adopt method 1, because method 1 is easier to understand.

The AudioGraph determines whether it allows users to playback the materials by comparing the user's input password with the password determined by the userID and Copy Protection record in the .aep file. In contrast, in the UNIX system allows a user with specific username to log on to the system by determining the result of comparing the user's encrypted password with the password kept in the file **/etc/passwd**.

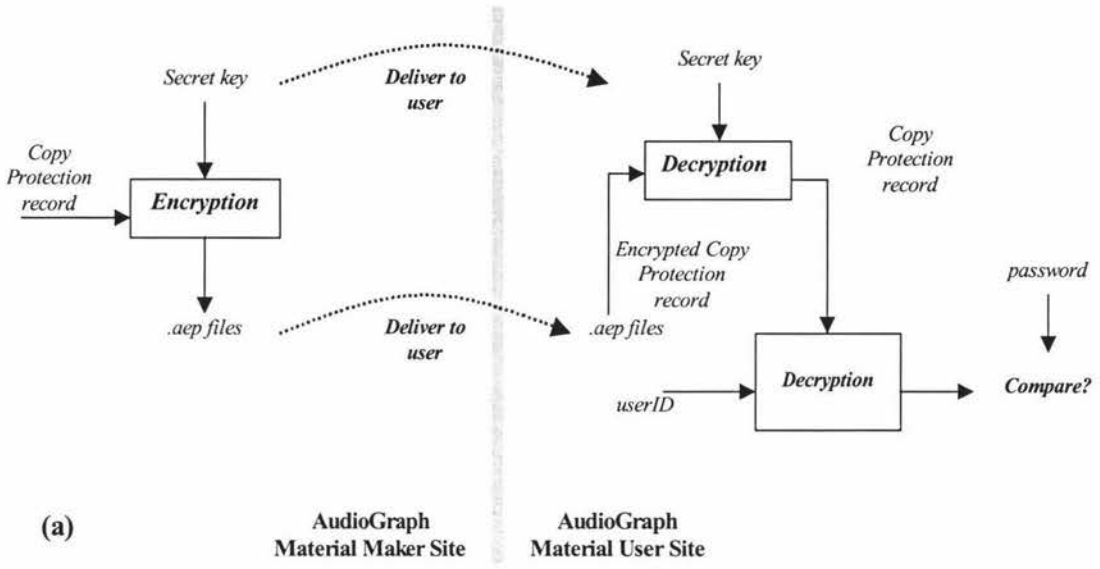
The Copy Protection key, which is used as key to encrypt the userID which is used as password, should be kept secret. The cryptanalysis or the opponent may already know the cryptographic algorithm, he/she may already know our method of usage protection, because it is relative easier for him/her to get information about what cryptographic algorithm and what approach is used in our AudioGraph Usage Control system. In this situation the only thing needed for him/her to totally crack our Usage Control system is the value of Copy Protection record. If we do not do anything about the Copy Protection record in the .aep file, then the cryptanalysis may be able to get the document of our .aep

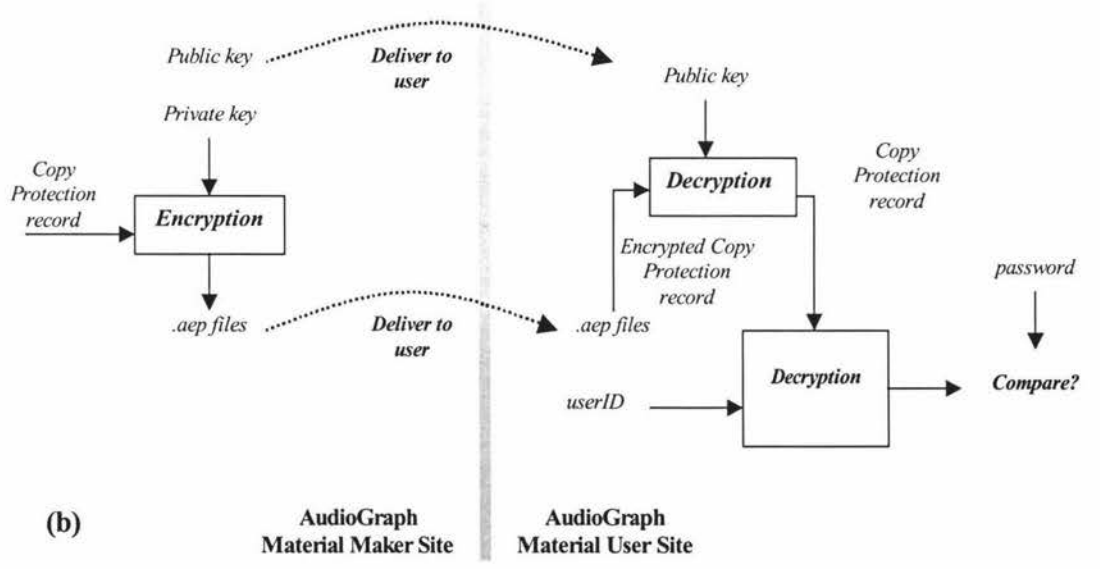
file format and read out the Copy Protection record somehow. This is a security threat to our system. There are three potential approaches to resolve this problem, as following:

1. ***Protect the contents of Copy Protection record by conventional cryptographic algorithm*** : Encrypt contents of Copy Protection record before inserting it into .aep file. Then when doing Usage Control in the AudioGraph Plug-in, it needs to decrypt the contents of Copy Protection record before it is used as the key to encrypt the userID. By this way, the contents of Copy Protection is hidden and the cryptanalysist or opponent can only get the ciphertext of the Copy Protection record from the .aep file. Figure 4.10 (a) illustrates this approach.
2. ***Protect the contents of Copy Protection record by public-key cryptographic algorithm***: Encrypt the contents of Copy Protection record using a private key before inserting it into the .aep files. Then, when doing Usage Control in AudioGraph Plug-in, it is necessary to decrypt the contents of the Copy Protection record using the public key before using it as the key to encrypt the userID. By this way, the contents of Copy Protection is hidden, the cryptanalysis or opponent can only get the ciphertext of the Copy Protection record from the .aep file. Figure 4.10 (b) illustrates this approach.
3. ***Protect the key to encrypt the userID by conventional cryptographic algorithm***: Insert the plaintext of Copy Protection record. When doing Usage Control in AudioGraph Plug-in, the contents of Copy Protection record is first encrypted, and then used in ciphertext format as the key to encrypt the userID. Therefore the key for the encryption of the userID has been protected even if the cryptanalysist or opponent knows the contents of the Copy Protection record in the .aep files. Figure 4.10 (c) illustrates this approach.

Method 1 requires both the Key Insertion Tool and AudioGraph Plug-in to embed the conventional encryption algorithm, and it requires synchronizing the secret key they use. Method 2 requires the Key Insertion Tool to embed the public-key encryption and decryption algorithm, and to know the private key for encryption, the AudioGraph Plug-in requires to know the public key for decryption; however this public key should not be known to the "public", it should be delivered to the user in a secure channel, otherwise the Copy Protection record would not get protected. Method 3 only requires the

AudioGraph Plug-in to embed the conventional cryptographic algorithm, and know the secret key for encryption of the contents of Copy Protection record.





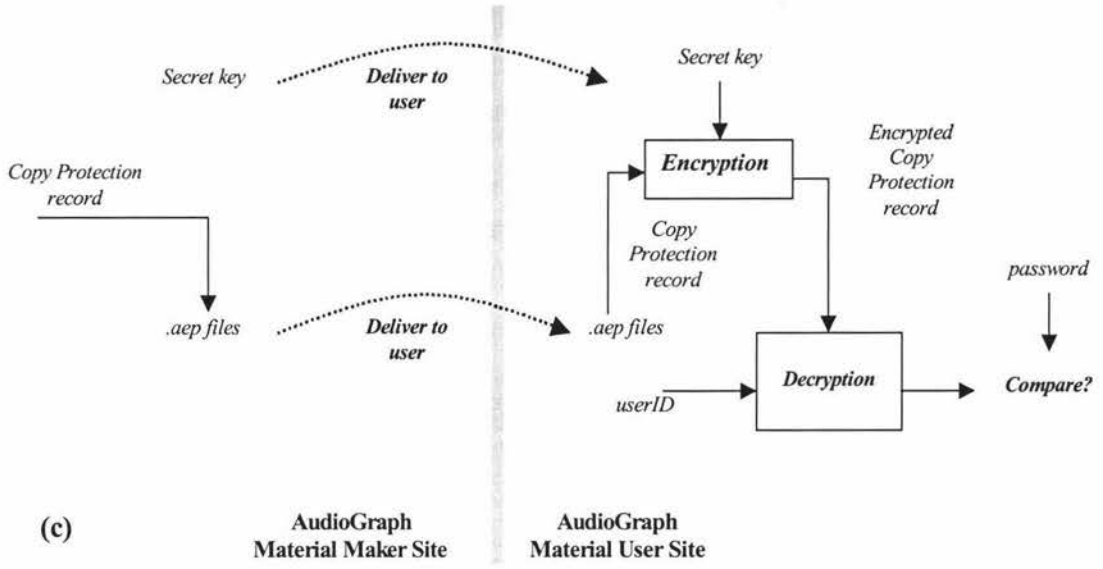


Figure 4.10: Approach of Key Protection

The advantage of method 1 and 2 are that they provide some degree of authentication, method 1 uses conventional encryption algorithm and method 2 uses public key encryption algorithm for the authentication. In Method 1 and Method 2, the AudioGraph Plug-in knows that the Copy Protection record is encrypted and inserted by the Key Insertion Tool, because the Key Insertion Tool is the only application that knows the corresponding key to encrypt the Copy Protection record. However, this authentication function maybe meaningless, for the reason that if the Company Key, Lower Protection Key and Upper Protection Key are all arbitrary bits sequences without any format that can be recognized, then after receiving the encrypted Copy Protection record in the .aep file, the AudioGraph Plug-in will not be able to tell if the Copy Protection record is encrypted and inserted correctly, because it can not tell whether the Copy Protection record has been correctly decrypted.

Compared with method 1 and 2, method 3 is a simpler and is still a secure approach. It does not need to synchronize the key between Key Insertion Tool and AudioGraph Plug-in. Everything is implemented inside the Usage Control part of the AudioGraph Plug-in. Although method 1 and method 2 have the advantage on the point of authentication, the Company Key, Lower Protection Key and Upper Protection are usually arbitrary bit sequences, and this makes the authentication void. Therefore, method 3 has been

adopted in the Usage Control of the AudioGraph Plug-in (also refer to Section 4.2.1 and Figure 4.8).

In the Usage Control part of AudioGraph Plug-in, there are a total of three secret keys: one is the secret key to encrypt the contents of Copy Protection record, then this encrypted Copy Protection record is used as a key to encrypt the userID. The other secret key is the key used to protect the userID/password on the local disk. The source of the key for encrypting the userID is from the Copy Protection record stored in the .aep file, and the contents of the Copy Protection record depends on the Company Key, Lower Protection Key and Upper Protection Key which are given by the AudioGraph producers and users. However, we need to create and manage the other two secret keys. There are two possible ways to manage these two keys:

1. Randomly choose two secret keys, then deliver them to the users in confidential channel, then users can install these two keys on their computer.
2. Just randomly choose two secret keys, then embed them into the AudioGraph Plug-in.

The first method is more flexible, whenever the keys need to be updated, new keys can just be chosen randomly and delivered to users. However there are drawbacks to this method. The key management is more complicated in first method than in second method. In the first method, the keys need to be managed, to be delivered and to be installed onto the user's computer, and this can be the target of the cryptanalysis or opponent. In the second method, whenever the keys need to be changed, new keys should be created and integrated into the AudioGraph Plug-in, then the new AudioGraph Plug-in will be built and be delivered to the users. The current AudioGraph security system adopts the second method; these two secret keys have been integrated into the AudioGraph Plug-in.

These two keys, especially the key to encrypt the contents of Copy Protection record, should be considered as the top secret of the AudioGraph security system. These two keys should not be updated too often for one good reason: whenever it is changeed, all the passwords that have been delivered to the users also need to be changed. That is to say, those users who were able to playback the material with the old Plug-in, will no longer be able to play them back again until they have all received a new password.

However, this is not a significant problem anyway, because the new password can be delivered to the users together with the new secret keys.

There is a disadvantage (a flaw) to this Usage Control system in the AudioGraph Plug-in: whenever a password for some material is exposed to the cryptanalysis or opponent, then there is no way to disable this userID/password pair. This flaw however, is also unavoidable if we use any other Usage Control scheme, it is inherent to this kind of Usage Control.

UserID and password

In the AudioGraph Security System, a userID can be randomly chosen by the user. However, unlike access control in a UNIX system and other multi-user operating system, which allow user to choose their favorite password, the AudioGraph Security System does not allow the user to choose a password for a specific userID. The password will be calculated by the system, based on userID and Copy Protection record. Therefore the password will be quite random in nature.

The advantage of a random password is that it is not guessable, it is nearly impossible by cryptanalysis or opponent to attack the system by guessing the password. But the user may have difficulty remembering it and so be temped to write it down, this may again raise the security risk. In general, a random password is poor in user acceptance. As mention earlier, the AudioGraph Security System stores the userID and password on the local disk as long as the user has entered a correct userID and password, therefore, the user only need to enter the userID and password once, this may increase the acceptance of a random password. Another way to increase the user acceptance is to use a minimum password length, without sacrificing the security of the system.

Considering the power of today's computer, a password should be at least 8 characters in length to protect against brute-force search. Most of the block cipher algorithms in commercial use have a block length of 64-bits, 128-bits or 256-bits. In AudioGraph Usage Control, the password is the output of encrypting a userID, therefore the length of the output may be either 64-bits, or 128-bits, or 256 bits. By converting this into a

character string using Radix-64 encoding (this will be discussed next chapter), the length of the password may be either 11-characters, 22-characters, or 43-characters in length, according to block size of the cryptographic algorithm. We decide to limit the length of password to 11-characters in the AudioGraph Usage Control, in order to increase user acceptance. In this case, the block size of the cryptographic algorithm used had better be 64-bits in length, although we can chop off extra bits if a cryptographic algorithm has more than a 64-bit block size, but this may increase the complexity of the algorithm, without increasing the degree of security.

4.2.3 Selection of the Encryption Algorithms

In previous section, the scheme of the AudioGraph Usage Control has been discussed. In the proposed Usage Control scheme, a conventional cryptographic algorithms will be embedded into the system. The cryptographic algorithm is the core part of the Usage Control system, it requires a good cryptographic algorithm.

There are three messages in the Usage Control required to be encrypted: the Copy Protection record, the userID and the userID/password pair stored in local disk file. Refer back to Figure 4.8, two different algorithms are used in our scheme, one is TEA encryption, another is the Blowfish encryption algorithm, the criteria and comparison of some cipher block algorithms will be given later in this section. Now we describe our decision for using these two cipher block algorithms instead of just one.

From the cryptanalyst's point of view, the task of breaking this Usage Control system is to deduce the key, which is used to encrypt the Copy Protection record. Let us have a look at the TEA algorithm first, a cryptanalysist can only get plaintext, i.e. a Copy Protection record. In this situation, his task is to get the ciphertext and deduce the TEA key, this task is similar to *ciphertext only attack* and this is the most difficult task in a cryptanalytic attack of an algorithm. Worse, he/she can only get a quite limited amount of plaintext, he/she may never be able to collect enough ciphertext in order to break the algorithm. Therefore, it is relative difficult to break the TEA algorithm, except if there is a breakthrough in the cryptanalysis of the TEA algorithm. For the Blowfish algorithm, the situation is not as good as for the TEA algorithm,. A cryptanalysist may be able to

collect some plaintext-ciphertext pairs (i.e., userID/password pairs). His/her task is therefore a *Known plaintext attack*, although this is still very difficult but it is easier than *ciphertext only attack*. Therefore a relative strong and yet more complicated cipher block algorithm – the Blowfish algorithm is used here. If a cryptanalyst wants to break our Usage Control system, he/she may start by collecting userID/password pairs, then try break the Blowfish algorithm first. If the cryptanalysis is very smart, and lucky enough to break off the Blowfish algorithm, then he/she can recover the Blowfish key.,, In this case he/she still does not know the TEA key used to encrypt the Copy Protection record, he/she may still have trouble to break the TEA algorithm in order to find out this TEA key, we are lucky to use different encryption algorithm here. If we use Blowfish to encrypt both the Copy Protection record and the userID, then in this situation, the cryptanalyst may be able to break the Blowfish algorithm again. Therefore we have good reason to use two different encryption algorithms instead of one. The other message that needs to be encrypted are the userID/password pairs, which will be stored on the local disk, however this is not very important, the simpler TEA algorithm is used, and it would be better to use a different TEA key here.

From above discussion, it is obvious that the Blowfish encryption plays a core role in the AudioGraph Usage Control System. The role of TEA encryption is also important but is less important than Blowfish algorithm, therefore we just simple choose the TEA algorithm to be used in our system here, as long as this algorithm has no serious flaw. We chose TEA because it is the simplest algorithm and it still remains unbreakable. Now we are going to explain our decision on the choice of Blowfish to be used in the AudioGraph Usage Control system.

Over the previous decades, there have been many conventional cryptographic algorithms used in commercial products over the world, many of them are very good and successful. To select the best and most suitable algorithm for our AudioGraph Security System is not going to be easy. The standard algorithm which has been used for long time is DES, however DES has nearly reached the end of its life, another good conventional algorithm is IDEA, however IDEA is a patented algorithm and it is not free for commercial use. In order to develop a new standard algorithm, the National Institute of Standards and Technology (NIST) has announced the initiation of the AES

(Advanced Encryption Standard) development effort (National Institute of Standards and Technology, 1997a). It has made a formal call for algorithms (National Institute of Standards and Technology, 1997b) on September 12, 1997. When we started to develop this project, in March of 1999, the NIST had gone to the second AES Candidate Conference, and selected five algorithms as finalists. These were **MARS** (IBM Research, 2001), **RC6** (RSA Security Inc., 2001), **Rijndael** (Rijmen, 1999; National Institute of Standards and Technology, 2001), **Serpent** (Anderson, 2001), **and Twofish** (Counterpane Internet Security Inc., 2001), the final selection of AES algorithm was not made until April 13-14, 2000. Although all of these five algorithms are very good, their block length are all over 128 bits (128 bits, 256 bits or 512 bits), which is not quite suitable for our need, we must therefore make a decision by ourselves.

In order to select a good cipher block algorithm, we need to investigate and compare some of the chosen algorithms. Our criteria of cryptographic algorithm selection is as following:

- *Security*: Security is the most important factor for a good cryptographic algorithm to be chosen for the AudioGraph security system. Security encompasses features such as resistance to cryptanalysis, soundness of its mathematical basis, randomness of the algorithm output, and relative security compared to other algorithm. A good cryptographic algorithm should be strong enough to be resistant against current known cryptanalysis techniques, it should not have weak keys or the number of weak keys should be a very trivial set in the key space.
- *Cost*: Cost is the second important criteria for a good cryptographic algorithm to be chosen for the AudioGraph security system. It encompasses licensing requirements, computational efficiency, etc. We need the algorithms used in AudioGraph security system to be free for commercial use. Speed is relative less important for our application, because the amount of data to be encrypted is very small, even if the speed is not fast enough, it will still meet our requirement.
- *Suitability and simplicity*: The cryptographic algorithm should be suitable to be used in AudioGraph security system. As mention earlier, an algorithm with block size of 64-bits is more suitable than with block size of 128-bits. A simpler algorithm is usually more computational efficiency and easier to be implement.

The job to compare various cryptographic algorithms is a difficult one, not only because the number of algorithms is large, but also because it requires extensive cryptographic knowledge and sound mathematical background, and even requires a good knowledge of cryptanalysis techniques. To actually investigate and compare various algorithms is beyond the scope of this thesis, we choose ca ryptographic algorithms used in AudioGraph security system according to above criteria and based on the result by other researcher's investigation and comparison.

It is hard to simple say which cryptographic algorithm is the best. Most of the investigations on cryptographic algorithms do a comparison on two aspects: practical comparison and parameters comparison. To do a practical comparison is to actually test the performance of the algorithm on the computer, they are usually used to compare the computational efficiency, memory requirement, speed etc. To do a parameter comparison means to compare some common parameters of the algorithm. Such parameters, like key length, rounds number, block length, are important to the algorithms. We will go through some of the result from other researchers and investigators, in order to judge our choice of Blowfish algorithm.

Table 4.2 shows the Comparative Block Ciphers Timings, most of the algorithms in Table 4.2 (Brown, 1998) have a 64-bits length block size (Square algorithm is the only one which has a 128-bits length block size). The algorithms have been implemented in Java and tested on Sun SPARC Solaris 1000 system. From this table we can see the speed of Blowfish algorithm is relatively good for the data encryption rate (617kbps).

Table 4.2: Comparative Block Ciphers Timings (Brown, 1998)

IJCE Timing Algorithm	Encryption (1MByte) Time (ms)	Encryption (1MByte) Rate (Kbps)	Key Init 1000 pairs (ms)
Blowfish	***13592***	***617***	***46536***
CAST5	15023	558	665
DES	26208	320	279
TripleDES	86904	96	971
IDEA	24696	339	408
LOKI91	15681	534	61
LOKI97 (+)	42340	198	1878
RC2	24073	348	553
RC4 (*)	8113	1033	1267
SAFER	34566	242	1323
Square	18580	451	1061

Table 4.3: Speed Comparisons of Block Ciphers on a Pentium (Counterpane Internet Security Inc., 1999a)

Algorithm	Clock cycles per round	# of rounds	# of clock cycles per byte encrypted	Notes
Blowfish	9	16	18	Free, unpatented
Khufu/Khafre	5	32	20	Patented by Xerox
RC5	12	16	23	Patented by RSA Data Security
DES	18	16	45	56-bit key
IDEA	50	8	50	patented by Ascom-Systec
Triple-DES	18	48	108	

Another practical comparison is available in (Counterpane Internet Security Inc., 1999a), the result is given in Table 4.3. This table shows the speed of the encryption algorithm by clock cycles number, the round numbers and the licensing information. Table 4.3 also shows us that the speed of Blowfish algorithm is the best compared with other block cipher in Table 4.3, it is also a free and unpatented algorithm.

Table 4.4 compares the key size of some cipher algorithms. It shows that the key size of Blowfish algorithm is variable. The advantage of cipher block algorithm with variable key size is that the user can use a longer key size to take against brute-force cryptanalysis. Remember that one of the reasons that DES can be broken by brute-force is that its key size is too short. Table 4.4 shows that the key size range of RC5 is better than Blowfish, but RC5 is a patented algorithm (see Table 4.3) and that does not meet our criteria of cost.

Table 4.4: The ESP CBC-Mode Cipher Algorithms (Pereira & Adams, 1998)

Algorithm	Key Size (bits)	Popular Key Size	Default Key Size
CAST-128	40 ~ 128	40, 64, 80, 128	128
RC5	40 ~ 2040	40, 128, 160	128
IDEA	128	128	128
Blowfish	40 ~ 448	128	128
3DES	192	128	128

Table 4.5 compares the performance of Twofish and Blowfish algorithms with other block cipher algorithms. It shows that the performance (speed) of Twofish and Blowfish are the best among those algorithms in the table. The Twofish algorithm is based on Blowfish algorithm, both algorithms are quite similar.

Table 4.5: Twofish - Performance vs. Other Block Ciphers (on a Pentium) (Counterpane Internet Security Inc., 1999b)

Algorithm	Key Length	Width (bits)	Rounds	Cycles	Clocks/Byte
Twofish	variable	128	16	8	18.1
Blowfish	variable	64	16	8	19.8
Square	128	128	8	8	20.3
RC5-32/16	variable	64	32	16	24.8
CAST-128	128	64	16	8	29.5
DES	56	64	16	8	43
Serpent	128, 192, 256	128	32	32	45
SAFER (S)K-128	128	64	8	8	52
FEAL-32	64, 128	64	32	16	65
IDEA	128	64	8	8	74
Triple-DES	112	64	48	24	116

From the above discussion, we can conclude that the Blowfish algorithm meets the criteria of *Cost* and *Suitability and simplicity*. Blowfish also meet the criteria of *Security*, since the Blowfish algorithm was published in April 1994, it has been examined by many cryptographers and, although some weak keys has been found and a 4-rounds variant of Blowfish has been broken, there have been no serious flaw found. It still remains a very strong algorithm, refer to (Counterpane Internet Security Inc., 2000; Schneier, 2001).

According to Counterpane Internet Security Inc., Blowfish has been used in over 140 products so far, and it is now standard in OpenBSD – a free multi-platform UNIX-like operating system, the list of products which use Blowfish are available from (Counterpane Internet Security Inc., 2002).

4.3 How Secure is the System

In Section 4.2, the proposed security system has been discussed in detail. From technical point of view, the strength of the system against cryptanalysis depends on the strength of the cryptography algorithm and the key length. A cryptanalysist may try all the possible way to break through the system – include brute-force attack and attack on the algorithm itself.

This section discusses the strength of the system again some of the most probable attacks on the system.

4.3.1 Brute-Force Attack on Keys and Password

Brute force cryptanalysis is the most straightforward cryptanalysis attack. The attacker uses raw computing power to search a key for the encryption of plaintext into ciphertext by trying all possible key combinations. Typically the attacker has either a ciphertext-plaintext pair or ciphertext only, whatever the attacker has, it is for him/her to find out a key and validate the correctness of the key. Typically, brute force attack is carried out either by using a collection of general-purpose computers or by using special-purpose hardware.

For the proposed AudioGraph security system, a cryptanalysis is possible to do brute-force attack by following two methods:

1. Attack on the password for a specific userID: that is to test all the possible combination of password for a specific userID.
2. Attack on Internal TEA key: Build a simulation program to simulate the AudioGraph security system illustrated in Figure 4.8, extract Copy Protection Record from aep file, and collect some userID/password pairs, then use Copy Protection Record and userID as input and test all the possible combination of Internal TEA key until the output of Blowfish encryption is exactly the same as password – then the value of Internal TEA key is found. After that find out the Internal TEA key and the system is totally broken.

Two parameters determine the speed of a brute-force attack: the number of keys to be tested and the speed of each test. Considering the attack on the password for a specific userID, a 64-bits password is converted into a 11-characters string password by using Radix-64 algorithm, each character can be one of the 64 different characters in the Radix-64 character set, therefore the number of possible passwords can be calculated as:

$$64^{11} \approx 7.378 \times 10^{19}$$

However, the attacker will have to try a password by typing it in them and waiting for the result from the AudioGraph Plug-in – all these will take at least 1 second, therefore, to test all the password will take about:

$$7.378 \times 10^{19} \text{ seconds} \approx 2.3 \times 10^{12} \text{ years}$$

This seems to be very long time. In other word, the chance for attacker to randomly input a correct password is very little.

To attack UNIX password protection system, an attacker may make a copy of file */etc/passwd* file and run a cryptanalysis program to attack on the password encryption. Because the cryptanalyst knows the algorithm to encrypt the password, he/she can design a simulation software to simulate the procedure of UNIX logon in, in this way each test of the key can be as fast as the power of computer allows it to be. However, it is impossible to simulate the AudioGraph security system. Refer back to Figure 4.8 on Section 4.2.1, it is obvious that if the Internal TEA key is not available, then it is impossible for the attacker to design simulation software to simulate the procedure of AudioGraph Usage Control.

Considering the brute-force attack on the Internal TEA key, the difficulty of brute-force attack depends on the length of TEA key. A TEA key is 128-bits length, i.e., there are 2^{128} possible keys in the TEA key space. Therefore to brute-force attack on the system will require to take 2^{128} attempt of the TEA key, with 50 percent chance of finding the correct key after half of the attempts. Assuming a supercomputer can try 100 million keys per second, it will take about 10^{23} years to find out the correct key among 2^{128} possible keys. Remember that the universe is only 10^{10} years old, and a supercomputer that can calculate 100 million keys per second would still be very powerful and expensive even in the following years, all these prove that a 128-bits TEA key would be long enough to defense against brute-force attack.

4.3.2 Attack On the Blowfish and TEA Encryption Algorithm

Attack On Blowfish Encryption Algorithm

To attack on a cryptographic algorithm, the attacker will have to find out a method to break the cryptosystem faster than breaking it by brute-force test. If it is worse than a brute-force test, then the method is worthless.

Since the Blowfish algorithm was published in April 1994, it has been examined by many cryptographers. They have tried to find out flaws and to exploit the flaws to build a cryptanalysis method to break the algorithm. However there have been no serious flaw found in the Blowfish algorithm, it still remains a very strong algorithm. Some of the attacks on Blowfish are the finding of weak keys and the cryptanalysis of less-round variants of the Blowfish. The weak keys of Blowfish has been examined by Serge Vaudenay (Vaudenay, 1996), these weak keys are a class of keys which can be detected – although not broken – in a 14 rounds or less variants of the Blowfish algorithm. The most advanced attack of the algorithms is the attack on the 4-rounds variants of the algorithm (Schneier, 2001), and that can not be extended to more rounds. It is reasonable to believe that Blowfish algorithm is a very strong and secure algorithm.

Attack On TEA Encryption Algorithm

The only attack on TEA encryption algorithm is a *related-key cryptanalysis* proposed by David Wagner (Kelsey et al.,1997). In related-key cryptanalysis, the cryptanalysts are supposed to have knowledge of the relationship between a pair of keys, as well as some data encrypted by these two keys, he/she does not know the keys themselves. In the chosen-plaintexts attack, a cryptanalyst get to choose the plaintext encrypted with the two keys. For TEA algorithm, it admits related-key attacks which arise from the severe simplicity of its key schedule. David Wagner discussed three related-key attack on the algorithm. The best attack can break TEA with just 2^{23} chosen plaintexts and one pair of related keys. In our AudioGraph security system, if key management is good enough, then the cryptanalyst will by no means get any information about the TEA key (this TEA key is the highest secret of our whole cryptosystem), the cryptanalyst can not even

access the ciphertexts, he/she can only get ciphertexts which is located in the Copy Protection record in .aep files, even so he/she may not be able to collect enough ciphertexts. Therefore it seems unrealistic for cryptanalysts to attack our system by related-key cryptanalysis of TEA algorithm.

4.3.3 Other possibly attack on the Security System

As mentioned in chapter 3, there are many strong algorithm out there and to choose a strong encryption algorithm is not easy, but the key management is harder. Key management is the hardest part in computer security. In the practical world, most of the attacks on cryptosystems are attacks on the key management, as it has proved to be more easier and cheaper than what??

In the previous chapter, two strong cryptography algorithms, i.e., Blowfish and TEA have been chosen and used in our cryptosystem. It seems fairly hard to break the cryptosystem by brute-force attack or attack on Blowfish and TEA encryption algorithms. However, it is possible for an attacker to break our cryptosystem by attacking the key management system.

The attacker may be able to steal the internal secret key by bribery to staff inside the company, or by intruding into the computer system in which the source code of the Key Insertion Tool, the secure AudioGraph Plug-in or the secret key are stored. The goal of the attacker is to get the internal secret key embedded inside the Key Insertion Tool or AudioGraph Plug-in. He/She can therefore get the internal secret key directly, or by debugging the source code of Key Insertion Tool and AudioGraph Plug-in. This would really be a nightmare to our AudioGraph security system.

The Key Insertion Tool is used to insert and check Copy Protection Record into ".aep" files, as well as generate userID/password pair, therefore this tool should be treated as very confidential asset as well. If the attacker get the copy of the tool somehow, he/she would be able to generate userID/password based on the Copy Protection Record inside ".aep" files, then he/she would be able to playback the presentation in the ".aep" files, or

even distribute the userID/password. In this case our security system is also totally broken.

The approach to defend against the attack on the key management is to have a good management of the key, the source code, the computer system on which the valuable asset are stored, and the management of the staff. The management of the keys, the source code and the computer system should ensure the security of all of these valuable assets. The management of the staff should ensure that the staff will not divulge any confidential information by accident or on purpose.

If the confidential information has been divulged, or our AudioGraph security system has been broken somehow, then the management should also be able to act to minimize the lost. The management issue is beyond the scope of this thesis and would not be discussed any further.

Chapter 5: Implementation of AudioGraph Security System

This chapter discusses the implementation of the AudioGraph security system in detail. The cryptography algorithms are described, the programs to implement the algorithm are explained. The method to transform the binary key into characters key string is also discussed in this chapter. At the end of this chapter the implementation of Key Insertion Tool and the Usage Control in the AudioGraph PC Plug-in are described.

5.1 The Blowfish Encryption Algorithm

5.1.1 The Algorithm

Blowfish block cipher algorithm was proposed by Bruce Schneier and published in 1994 (Schneier, 1994; Counterpane Internet Security Inc., 2000). The purpose of designing this algorithm is to let the world have a secure, unpatented and free-available encryption algorithm, and to replace the insecure DES algorithm. Since its invention, it is slowly being accepted by many people around the world.

Blowfish is a secret-key block cipher with a 64-bit data block and a variable-length key length. The key could be up to 448-bits length. The architecture of the algorithm is illustrated in Figure 5.1. Similar to DES algorithm, Blowfish is a 16-round Feistel network, each round consists of a key dependent permutation, and a key and data dependent substitution.

Blowfish uses a subkey array called P-array, which consists of eighteen 32-bit length of subkeys:

$$P_1,\ P_2,\ \ldots,\ P_{18}$$

There are four 32-bit S-boxes used in Blowfish, each S-box has 256 entries:

$$S_{1,0},\ S_{1,1},\ \ldots,\ S_{1,255}$$

$$S_{2,0},\ S_{2,1},\ \ldots,\ S_{2,255}$$

$$S_{3,0},\ S_{3,1},\ \ldots,\ S_{3,255}$$

$$S_{4,0},\ S_{4,1},\ \ldots,\ S_{4,255}$$

These P-array and S-boxes are calculated dependent on the Blowfish key. The method of calculation on them will be described later.

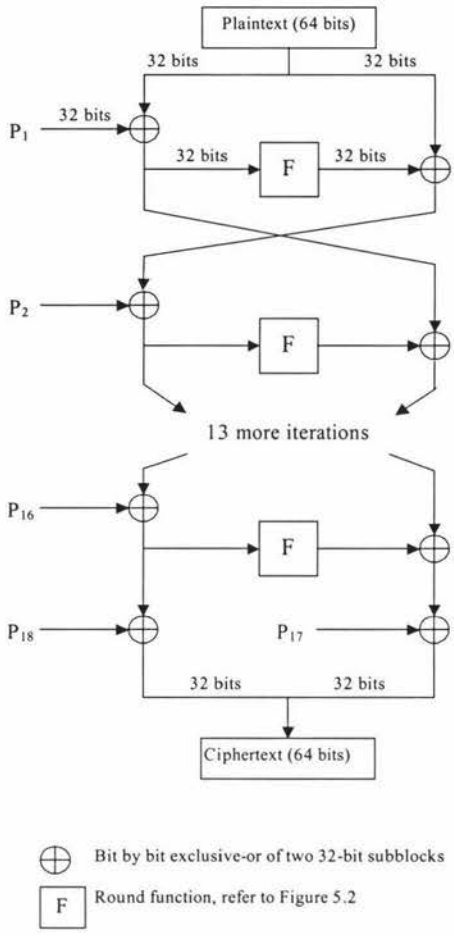


Figure 5.1: Block Diagram of Blowfish Block Cipher Algorithm (Schneier, 1994; Counterpane Internet Security Inc., 2000)

Encryption (refer to Figure 5.1)

1. Divide input into two 32-bit halve: L and R
2. For i = 1 to 16 do

   $L = L \oplus P_i$

   $R = F(L) \oplus R$

   swap L and R
3. swap L and R
4. $R = R \oplus P_{17}$

   $L = L \oplus P_{18}$
5. Recombine L and R

The procedure of decryption is the same as encryption, except that the P-Array P_1, P_2, … P_{18} are used in the reverse order.

Round Function F

Figure 5.2 illustrates the structure of round function F, it takes a 32 bit input and divides it into four 8-bit quarter. Each quarter works as index into each S-box, the output is a 32-bit sub-block. Supposed the outputs from each S-boxes are S_1, S_2, S_3 and S_4, then the output of round function is:

$$F(x) = ((S_1+S_2 \bmod 2^{32}) \oplus S_3)+S_4 \bmod 2^{32}$$

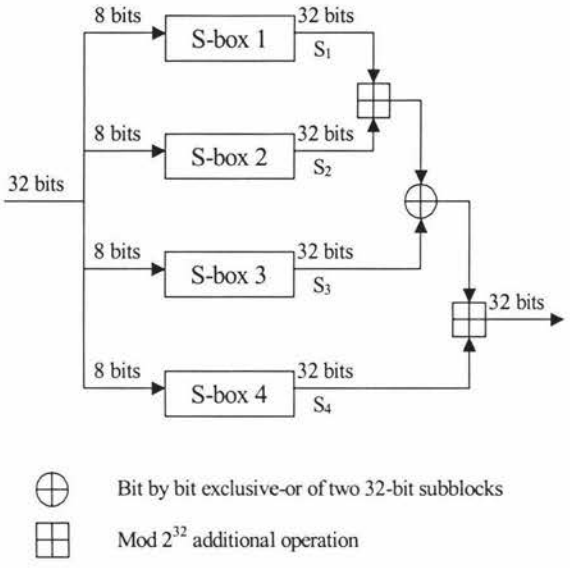


Figure 5.2: Round Function F in Blowfish Algorithm (Schneier, 1994; Counterpane Internet Security Inc., 2000)

Initializing the P-Array and S-Boxes

The P-array and S-boxes depend on the Blowfish key. The procedure to calculate them is as follow:

1. Initialize the P-array with fixed string:

```
P1 = 0x243f6a88
P2 = 0x85a308d3
P3 = 0x13198a2e
P4 = 0x03707344
```

2. XOR P_1 with the first 32-bits of the Blowfish key, XOR P_2 with the second 32-bits of the key, and so on until P_{14}. Then cycle through the first 32-bits of key, i.e., XOR P_{15} with first 32-bits of key, repeat until P_{18}.
3. Now there are values for subkey P_1, P_2, …, P_{18}. We can encrypt an all-zero string with the Blowfish algorithm. Replace the P_1 and P_2 with this output.
4. Encrypt the output with the Blowfish algorithm using the modified subkeys. Replace P3 and P4 with this output.
5. Continue the process, until all the entries of the P-array and all entries of the four S-Boxes have been replaced by the output of the continuously-changing Blowfish algorithm.

Compared with DES algorithm, the S-box of Blowfish has two main features, which make it more secure than DES algorithm:

1. Large S-boxes: The S-boxes of DES has 8 × 64 entries, the output of each entries is 4-bits long; in comparison, the S-boxes of Blowfish has 4 × 258 entries, the output of each entries is 32-bits long. A larger S-boxes are more resistant to *differential cryptanalysis*.
2. Key-dependent S-boxes: The S-boxes of DES have fixed value. A fixed S-boxes must be designed to be resistant to *differential* and *linear cryptanalysis*, however the S-boxes of DES had been designed to be resistant to *differential cryptanalysis*, but it had not been designed to be resistant to *linear cryptanalysis*. The S-boxes of Blowfish is key-dependent, it is much more resistant to these attacks.

5.1.2 The Program

In our AudioGraph security system, it only requires to perform Blowfish encryption. When doing Blowfish encryption, three related functions have been defined:

- `blowfishEncrypt( BLOWFISH_KEY *key, BYTE *data )`: The function to encrypt data. The pointer "`data`" point to the buffer of data being encrypted, and the pointer "`key`" point to a Blowfish key used to encrypted the data.
- `blowfishKeyInit(BLOWFISH_KEY *key, BYTE *userKey, int userKeyLength)`: The function to initialize the P-array and S-boxes. Parameter

"key" is a pointer to the Blowfish key, which consists of P-array and S-boxes. Parameter "userKey" is a pointer to a user key, and parameter "userKeyLength" is the byte number of the user key.

- `initSBox(BLOWFISH_KEY *key, LONG *Sbox, BYTE *buffer)`: The function to initialize the S-boxes.

The implementation of the functions is not very complicated, they are illustrated below:

1. Function `blowfishEncrypt( BLOWFISH_KEY *key, BYTE *data ):`

```
/*****************************/
/*   Do Blowfish Encryption  */
/*****************************/
void blowfishEncrypt( BLOWFISH_KEY *key, BYTE *data )
{
      BYTE *dataPtr = data;
      LONG *P = key->P, *S1 = key->S1, *S2 = key->S2;
      LONG *S3 = key->S3, *S4 = key->S4;
      LONG L, R;

      L = mgetBLong( dataPtr );
      R = mgetBLong( dataPtr );

      /* Perform 16 rounds of encryption */
      oddRoundE(   1, P, S1, S2, S3, S4 );
      evenRoundE(  2, P, S1, S2, S3, S4 );
      oddRoundE(   3, P, S1, S2, S3, S4 );
      evenRoundE(  4, P, S1, S2, S3, S4 );
      oddRoundE(   5, P, S1, S2, S3, S4 );
      evenRoundE(  6, P, S1, S2, S3, S4 );
      oddRoundE(   7, P, S1, S2, S3, S4 );
      evenRoundE(  8, P, S1, S2, S3, S4 );
      oddRoundE(   9, P, S1, S2, S3, S4 );
      evenRoundE( 10, P, S1, S2, S3, S4 );
      oddRoundE(  11, P, S1, S2, S3, S4 );
      evenRoundE( 12, P, S1, S2, S3, S4 );
      oddRoundE(  13, P, S1, S2, S3, S4 );
      evenRoundE( 14, P, S1, S2, S3, S4 );
```

```
        oddRoundE(  15, P, S1, S2, S3, S4 );
        evenRoundE( 16, P, S1, S2, S3, S4 );

        /* Perform the final XOR's */
        L ^= P[ 16 ];
        R ^= P[ 17 ];

        dataPtr = data;
        mputBLong( dataPtr, R );
        mputBLong( dataPtr, L );
}
```

This function performs the Blowfish encryption of data block. Inside the function, the data block is separated into two halves – L and R, then these two halves go through sixteen iterations operation by calling to a macro function `oddRoundE` for odd iteration and `evenRoundE`, inside which the subkey (P-array element) are applied. The last two subkeys (`P[16]` and `P[17]`) are then applied to exclusive-or with the output of the 16th iteration. Then the two halves are combined as output, which is a 64-bit length of encrypted data block.

In the function, `BLOWFISH_KEY` is a structure which contains the P-array and the four S-boxes as member variables, as defined below:

```
typedef struct {
        LONG  P[ BLOWFISH_PARRAY_SIZE ];          /* P-array */
        LONG  S1[ BLOWFISH_SBOX_SIZE ],
              S2[ BLOWFISH_SBOX_SIZE ],
              S3[ BLOWFISH_SBOX_SIZE ],
              S4[ BLOWFISH_SBOX_SIZE ];           /* S-boxes */
} BLOWFISH_KEY;
```

The macro function `mgetBLong(dataPtr)` will combine 4-bytes in `dataPtr` and return a 32-bit long integer. Macro function `mputBLong( dataPtr, L )` will take a 32-bit long integer `L` and separate it into 4-byte, `dataPtr` is a pointer to the result. They are defined as following:

```
#define mgetBLong(memPtr)
            ( ( ( LONG ) memPtr[ 0 ] << 24 ) | \
              ( ( LONG ) memPtr[ 1 ] << 16 ) | \
              ( ( LONG ) memPtr[ 2 ] << 8 )  | \
                ( LONG ) memPtr[ 3 ] );
            memPtr += 4

#define mputBLong(memPtr,data)
            memPtr[ 0 ] = ( BYTE ) ( ( ( data ) >> 24 ) & 0xFF );
            memPtr[ 1 ] = ( BYTE ) ( ( ( data ) >> 16 ) & 0xFF );
            memPtr[ 2 ] = ( BYTE ) ( ( ( data ) >> 8 ) & 0xFF );
            memPtr[ 3 ] = ( BYTE ) ( ( data ) & 0xFF );
            memPtr += 4
```

Function `oddRoundE` and `evenRoundE` are the iteration operation for odd and even iteration respectively, they are defined as:

```
#define oddRoundE(count,P,S1,S2,S3,S4)     L ^= P[ count - 1 ];
                                           R ^= f( L, S1, S2, S3, S4 )
#define evenRoundE(count,P,S1,S2,S3,S4)    R ^= P[ count - 1 ];
                                           L ^= f( R, S1, S2, S3, S4 )
```

Here the round function `f` is defined as:

```
/*********************/
/* The f-function */
/*********************/
#define exta(x)          ((int)((x>>24)&0xFF))
#define extb(x)          ((int)((x>>16)&0xFF))
#define extc(x)          ((int)((x>>8)&0xFF))
#define extd(x)          ((int)((x)&0xFF))
#define f(data,S1,S2,S3,S4)
      (((S1[exta(data)]+S2[extb(data)])^S3[extc(data)])+S4[extd(data)])
```

2. Function ***blowfishKeyInit(BLOWFISH_KEY *key,***
BYTE *userKey,
int userKeyLength):

```
/*************************/
/* Set up a Blowfish key */
/*************************/
int blowfishKeyInit( BLOWFISH_KEY *key, BYTE *userKey,
                     int userKeyLength )
{
      BYTE buffer[ BLOWFISH_BLOCKSIZE ];
      int keyIndex = 0, i;

      /* Set up the initial P-array and
         S-boxes based on the digits of pi */
      memcpy( key->P, initialParray, sizeof( initialParray ) );
      memcpy( key->S1, initialSbox1, sizeof( initialSbox1 ) );
      memcpy( key->S2, initialSbox2, sizeof( initialSbox2 ) );
      memcpy( key->S3, initialSbox3, sizeof( initialSbox3 ) );
      memcpy( key->S4, initialSbox4, sizeof( initialSbox4 ) );

      /* XOR the user key bits into the P-array */
      for( i = 0; i < BLOWFISH_NO_ROUNDS + 2; i++ )
      {
            LONG value = 0L;  /* Needed for > 32-bit processors */
            int byteIndex;

            /* Get 32 bits of user key and XOR them into the P-array */
            for( byteIndex = 0; byteIndex < 4; byteIndex++ )
            {
                  value = ( value << 8 ) | userKey[ keyIndex++ ];
                  keyIndex %= userKeyLength;
            }
            key->P[ i ] = key->P[ i ] ^ value;
      }

      /* Encrypt the all-zero string with the initial P-array
         to get the final P-array */
      memset( buffer, 0, BLOWFISH_BLOCKSIZE );
      for( i = 0; i < BLOWFISH_NO_ROUNDS + 2; i += 2 )
```

```
    {
        BYTE *bufferPtr = buffer;

        blowfishEncrypt( key, buffer );
        key->P[ i ] = mgetBLong( bufferPtr );
        key->P[ i + 1 ] = mgetBLong( bufferPtr );
    }

    /* Continue the process to fill the S-boxes */
    initSBox( key, key->S1, buffer );
    initSBox( key, key->S2, buffer );
    initSBox( key, key->S3, buffer );
    initSBox( key, key->S4, buffer );

    return( CRYPT_OK );
}
```

This function sets up and initializes the Blowfish key. It starts to initialize the P-array and S-boxes with initial constant by copy them into the key data structure, then the P-array of the Blowfish key is exclusive-or with the user key bit. Then it goes to set up the value of the P-array by replace two elements each time with the encrypted data output using the continuous changing key. After all elements of the P-array have been set up, it then goes to set up the four S-boxes by calling the function initSbox() (will be described later).

In this function, the parameter userKey is the user key which is in string format, it can be up to 448-bits length and is used to initialize the Blowfish key. The parameter userKeyLength is the byte number of the userKey.

3. Function `initSBox( BLOWFISH_KEY *key, LONG *Sbox, BYTE *buffer ):`

```
/**************************/
/* Set up a Blowfish S-box */
/**************************/
static void initSBox( BLOWFISH_KEY *key, LONG *Sbox, BYTE *buffer )
{
    int sBoxIndex;
```

```
        for( sBoxIndex = 0; sBoxIndex < 256; sBoxIndex += 2 )
        {
                BYTE *bufferPtr = buffer;

                blowfishEncrypt( key, buffer );
                Sbox[ sBoxIndex ] = mgetBLong( bufferPtr );
                Sbox[ sBoxIndex + 1 ] = mgetBLong( bufferPtr );
        }
}
```

This function is used to initialize the S-boxes of the Blowfish key. All the 256-entries in each of four S-boxes need to be initialized.

Here, parameter `key` is a pointer to the Blowfish key; parameter `Sbox` is a pointer to one of the four S-Boxes in Blowfish algorithm, each S-Box contains 256-entries; parameter `buffer` is a pointer to an 8-byte buffer, this buffer is used to store the data block.

5.2 The TEA Encryption Algorithm and Program

The Tiny Encryption Algorithm (TEA) was developed by David Wheeler and Roger Needham at the Computer Laboratory of Cambridge University (Wheeler & Needham, 1994). It is a Feistel Network block cipher which uses operations from mixed (orthogonal) algebraic groups - XORs and additions. It encrypts a 64-bit data block using a 128-bit key.

Although each nonlinear iteration operation in this Feistel Network block cipher is weak, it archives strong security by iteration it many round. After six round it achieves complete diffusion – a single bit change in data block or key has spread very close to 32 bit change. The algorithm developer considered a 16-round algorithm would suffice, and it is suggest to use 32-rounds in the TEA algorithm.

The TEA encryption algorithm is extremely simple, it can be described by C programming language in just a few lines (Wheeler & Needham, 1994), as illustrated below:

```
#define TEA_ROUNDS 32
void tea_code(tea_block *v, tea_key *k)
{
      unsigned long y=v->v[0], z=v->v[1], n=TEA_ROUNDS;
      unsigned long delta=0x9e3779b9, sum=0;
      while(n-->0){
            sum += delta;
            y += ((z<<4)+k->k[0]) ^ (z+sum) ^ ((z>>5)+k->k[1]);
            z += ((y<<4)+k->k[2]) ^ (y+sum) ^ ((y>>5)+k->k[3]);
      }
      v->v[0] = y;
      v->v[1] = z;
}
```

The C function `tea_code(tea_block *v, tea_key *k)` implements TEA encryption. It first separates the data block into two halves – `y` and `z`, then it apply the TEA key to the iteration operation over `y` and `z` for 32 rounds. The output `y` and `z` from the 32th round are combined together as output encrypted data block.

In this function, `tea_block` is the data structure to represent the data block for the TEA algorithm, and is defined as:

```
typedef struct tea_block{
      long v[2];
}tea_block;
```

The `tea_key` is the data structure to represent a TEA key, it is defined as:

```
typedef struct tea_key{
        long k[4];
}tea_key;
```

The decryption procedure is exactly the reverse of the encryption procedure, following C code describes this TEA decryption procedure (Wheeler & Needham, 1994):

```
void tea_decode(tea_block *v, tea_key *k){
      unsigned long y=v->v[0], z=v->v[1], n=TEA_ROUNDS;
```

```
        unsigned long delta=0x9e3779b9, sum;
        sum = delta << 5;
        while(n-->0){
                z -= ((y<<4)+k->k[2]) ^ (y+sum) ^ ((y>>5)+k->k[3]);
                y -= ((z<<4)+k->k[0]) ^ (z+sum) ^ ((z>>5)+k->k[1]);
                sum -= delta;
        }
        v->v[0] = y;
        v->v[1] = z;
}
```

Like DES, all the block cipher can be used in a cryptosystem in four different operation modes: *Electronic Codebook (ECB) Mode*, *Cipher Block Chaining (CBC) Mode*, *Cipher Feedback (CFB) Mode* and *Output Feedback (OFB) Mode*. In our AudioGraph security system the TEA algorithm is used in *ECB* mode. In the *ECB* mode, the plaintext is handled one block at a time, each block of plaintext is encrypted to the same block of ciphertext using the same key. The following C code implements the *ECB* operation mode for TEA algorithm:

```
void tea_ecb(tea_block *data, tea_key *key, int len, int encrypt){

        tea_block bk;
        if(encrypt){
                for(int i=0; i<len; i++){
                        bk.v[0] = data[i].v[0];
                        bk.v[1] = data[i].v[1];
                        tea_code(&bk, key);
                        data[i].v[0] = bk.v[0];
                        data[i].v[1] = bk.v[1];
                }
        }
        else{
                for(int i=0; i<len; i++){
                        bk.v[0] = data[i].v[0];
                        bk.v[1] = data[i].v[1];
                        tea_decode(&bk, key);
                        data[i].v[0] = bk.v[0];
                        data[i].v[1] = bk.v[1];
                }
```

```
        }
}
```

Here the parameter `data` is a pointer to the TEA data block (or a number of TEA data blocks), parameter `key` is a TEA key, parameter `len` is the number of data block, parameter `encrypt` specify whether it is going to do encryption (if `encrypt!=0`) or decryption (if `encrypt==0`).

5.3 Conversion Between Arbitrary Bit Stream and Printable Characters

In our AudioGraph security system, the method to verify if a userID and password is correct or not is to compare the encrypted userID with the password, a correct password should be exactly the same as its encrypted userID (refer back to Figure 4.8 and Figure 4.9). However, an ASCII character string of the userID from the user's input may become a stream of arbitrary 8-bit bytes after being encrypted. In order to compare the user input password with the encrypted userID, the encrypted userID should be converted into a stream of printable ASCII characters.

In principle, any printable encoding scheme to convert a stream of arbitrary 8-bit bytes into printable ASCII characters would be suffice, since it would not change the underlying binary bit stream. A scheme referred to as Radix-64 conversion, which is used in both PGP and PEM to transfer encrypted email over the network, is chosen in our AudioGraph security system to serve this purpose.

5.3.1 Radix-64 Encoding

In the Radix-64 encoding scheme, a 24-bit groups of input bits is divided into four 6-bit sets, each 6-bit set is mapped into a character. Table 5.1 shows the mapping of 6-bit input values to characters. The character set consists of the alphanumeric characters plus "+" and "/". The "=" character is used as the padding character.

6-bit Value	Character Encoding	6-bit Value	Character Encoding	6-bit Value	Character Encoding	6-bit Value	Character Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Table 5.1: Radix-64 Encoding

By using Radix-64 encoding, it is possible to convert any arbitrary bit stream into ASCII character string that is "immune" to the modifications inflicted by mail systems, so it is adopted by PGP and PEM to transfer encrypted email or encrypted data file. For more information about PGP and Radix-64 conversion, please refer to (Atkins, 1991).

5.3.2 Encoding of userID

As discussed in previous chapter, the userID is restricted to be 8 characters long. In the AudioGraph Usage Control in Plug-in, this userID will be encrypted using Blowfish algorithm. The encrypted userID is a 64-bit length data block. It requires to be converted into printable characters using Radix-64 encoding.

In our AudioGraph security system, this 64-bit data block is divided into 11 6-bit section (the last section contains two 0 padding bits), each 6-bit section is then encoding into a character according to Table 5.1. Therefore the encrypted userID is converted into a 11-character string, and it is the correct password for the corresponding userID.

The following 'C' code implements the encoding of userID.

```
void userIDEncoding(BYTE *input, BYTE *output)
{
      long inputL = 0L, inputR = 0L;
```

```
BYTE buffer[11];          // to keep the temporary 6-bit set
                          // from input

BYTE *bytePtr = input;    // input is a 8-byte length
                          // arbitrary bit stream

inputL = mgetBLong(bytePtr);  // convert 4-byte of input into a
                              // long integer
inputR = mgetBLong(bytePtr); // convert another 4-byte of input
                              // into a long integer

// convert the input bit stream into 6-bits set,
// and store them in the buffer
buffer[ 0 ] = ( BYTE ) (inputR & 0x3F );
cipherR = cipherR >> 6;
buffer[ 1 ] = ( BYTE ) (inputR & 0x3F );
cipherR = cipherR >> 6;
buffer[ 2 ] = ( BYTE ) (inputR & 0x3F );
cipherR = cipherR >> 6;
buffer[ 3 ] = ( BYTE ) (inputR & 0x3F );
cipherR = cipherR >> 6;
buffer[ 4 ] = ( BYTE ) (inputR & 0x3F );
cipherR = cipherR >> 6;

buffer[ 5 ] = ( BYTE )( ( (inputL &0x0F) << 2) \
                        | (inputR &0x03) );
cipherL = cipherL >> 4;

buffer[ 6 ] = ( BYTE ) (inputL & 0x3F );
cipherL = cipherL >> 6;
buffer[ 7 ] = ( BYTE ) (inputL & 0x3F );
cipherL = cipherL >> 6;
buffer[ 8 ] = ( BYTE ) (inputL & 0x3F );
cipherL = cipherL >> 6;
buffer[ 9 ] = ( BYTE ) (inputL & 0x3F );
cipherL = cipherL >> 6;
buffer[ 10 ] = ( BYTE ) (inputL & 0x0F );

//Radix-64 encoding
```

```
        for(int i=0; i<11; i++){
                //only 4 bits are from cipher text
                output[i] = (BYTE) ( radixEncode(buffer[i]) );
        }
}
char radixEncode(int bitValue)
{
        return radixArray[bitValue];
}
const char radixArray[64] = {
        'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', \
        'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', \
        'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', \
        'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', \
        'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', \
        '8', '9', '+', '/' };
```

5.4 Integrated the Algorithms into AudioGraph PC Plug-in

In the implementation of AudioGraph security system, a data structure is defined to store the Copy Protection record (description in Section 4.2.2) in the memory, as following:

```
//=============================================
// Class RKey
//=============================================
class RKey: public RRecord
{
public:
        long m_companyKey;
        long m_lowerProtectionKey;
        long m_upperProtectionKey;

        RKey(long companyKey, long lowerProtectionKey, \
                long upperProtectionKey):
                RRecord(T_Key)
        {
                this->m_companyKey = companyKey;
                this->m_lowerProtectionKey = lowerProtectionKey;
                this->m_upperProtectionKey = upperProtectionKey;
```

```
    }
};
```

When the Web browser encounters a tag like `<EMBED SRC = "xxx.aep">` in the .html file, it will start to initialize the AudioGraph Plug-in by invoking method "NP_Initialize" and "NP_GetEntryPoints" in the AudioGraph Plug-in Dynamic Link Library (DLL). Inside the "NP_GetEntryPoints" definition, more plug-in entry points are defined, those entry points are actually Plug-in implemented methods, they are use to further initialize the Plug-in and process the data stream. One of these methods is "*NPP_Write(NPP instance, NPStream *stream, int32 offset, int32 len, void *buffer)*", this will deliver a block of data from .aep file to the AudioGraph Plug-in.

In definition of method "*NPP_Write*", other methods is invoked to treat the data block. This data block will be divided into various AudioGraph records. On encountering the Copy Protection record, it will create a "RKey" object, and extract the Company Key, Lower Protection Key and Upper Protection Key from data block keep them in the "RKey" object, as described in following code:

```
void CUserWnd::ProcessRecord() {
    ......
    switch( type ) {
    ......
    case 'T':   // Copy Protection Record
        {
            long companyKey = ser->ReadLong();
            long lowerProtectionKey = ser->ReadLong();
            long upperProtectionKey = ser->ReadLong();
            rec = new RKey(companyKey, lowerProtectionKey,
                                upperProtectionKey);
        }
        break;
    ......
    }
    if( rec != NULL ) {
        mRecordList.AddTail( rec );
        ......
    }
```

```
        ......
}
```

In the implementation of AudioGraph Plug-in, all records are stored in a double link-list. A thread is created inside AudioGraph Plug-in to playback the presentation, the thread will go through the link-list and playback the records one by one. When it goes to the RKey object, it will perform the security check by calling methods "*CUserWnd::OnKeyMsg()*" and "*CUserWnd::DoEncryption()*". The Blowfish encryption, TEA encryption and Radix-64 conversion are integrated into these methods. On execution of method "*CUserWnd::OnKeyMsg()*", it will try to load the userID and password from a Hard Disk file. If it succeed in loading the userID and password from the Hard Disk file, it will then check if the password is correct (the mechanism of security check in AudioGraph Plug-in has already been illustrated in Figure 4.9). If the password is correct, the thread will go on to the next record in the link-list. If the password from Hard Disk is not correct, or if it can not find the userID and password pair from the Hard Disk file, a dialog window like Figure 5.3 will appear:

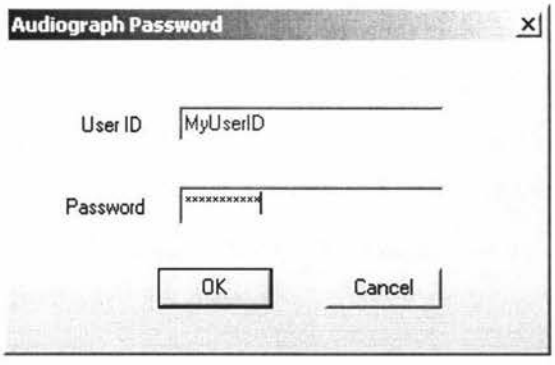


Figure 5.3: userID and password Dialog Window

Then the user may input his/her userID and password and click on button "OK", and the security system will do check if the password is correct. If it pass the security check, the thread will go to next record in the link-list and continue to playback the remaining records in the linked-list; otherwise a dialog window, similar to Figure 5.4 will appear and display warning message, and the playback procedure will be stopped and disabled.

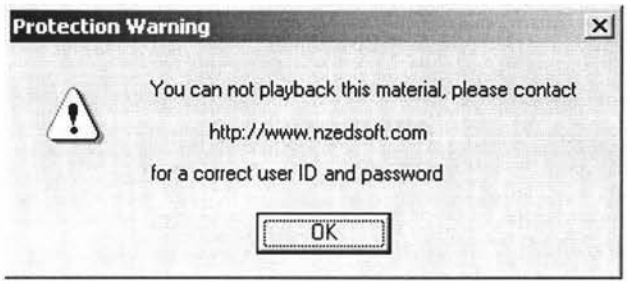


Figure 5.4: Protection Warning Dialog Window

The source code of methods "*CUserWnd::OnKeyMsg()*" and "*CUserWnd::DoEncryption()*" can be found in Appendix A.

5.5 Implementation of Key Insertion Tool

Key Insertion Tool has been implemented as a standard independent application. The main interface is illustrated in Figure 5.5.

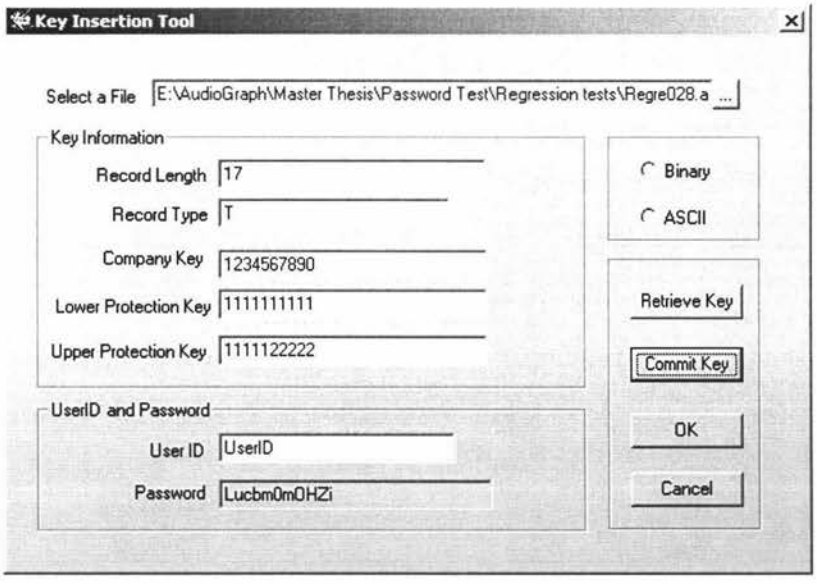


Figure 5.5: Key Insertion Tool

The implementation of Key Insertion Tool allows user to specify a .aep file name or select a .aep file from the local hard disk. As soon as the user has specified an .aep file, it will check if there is a *Copy Protection Record* inside that .aep file. If the .aep file contains a *Copy Protection Record*, it will retrieve the *Company Key*, the *Lower Protection Key* and the *Upper Protection Key* from this *Copy Protection Record* and

display them in the corresponding text box, as illustrated in Figure 5.5. Otherwise it will display a message telling the user that it does not contain a *Copy Protection Record*. The user can change the current *Company Key*, the *Lower Protection Key* and *Upper Protection Key*, or specify new value for *Company Key*, the *Lower Protection Key* and the *Upper Protection Key*. After the user click on button "Commit Key", these *Company Key*, *Lower Protection Key* and *Upper Protection Key* will be updated or inserted into the specified .aep file.

The Record Length and Record Type section in above interface is used to specify the length and type of the *Copy Protection Record*. According to the AudioGraph File Format, the length should always be a integer 17 and the type should always be a character 'T' for the *Copy Protection Record*.

The function to determine value of a password for the specified userID is also integrated into this Key Insertion Tool. Refer to Figure 5.5, we can find out the value of password for a specified userID. Whenever the value of userID is changed, its corresponding value of password would be changed as well. The mechanism to calculate the password for a specified userID is illustrated in Figure 5.6. The *Internal TEA key* should be the same as in Usage Control inside AudioGraph Plug-in (refer back to Figure 4.9).

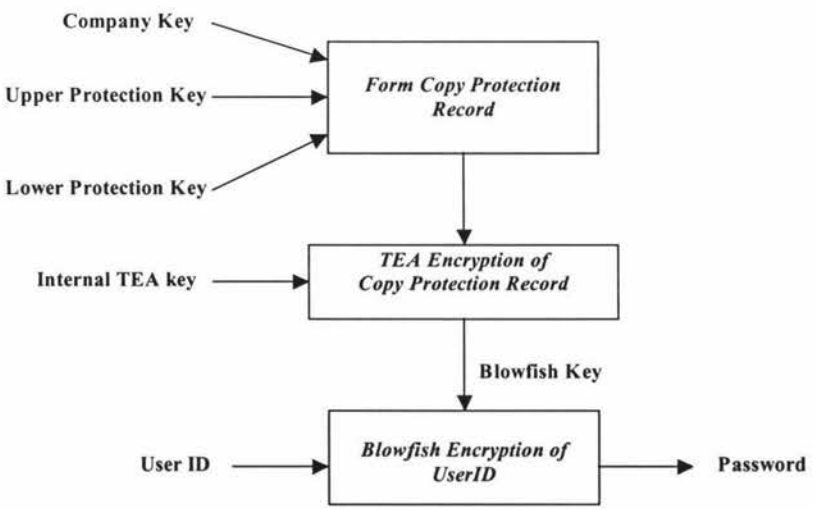


Figure 5.6: Calculation of Password for a Specified UserID

The source code to calculate the password for a specified userID can be found in Appendix B. In the implementation, it also does TEA decryption and compares the result with *Copy Protection Record* to verify the correctness of TEA encryption, and it does Blowfish decryption and compares the result with original UserID to verify the correctness of the password. Interesting reader can refer to Appendix B for more details.

Chapter 6: Testing

Software testing encompasses many of the activities refereed to as software quality assurance (SQA). The purpose of software testing is to execute a program with the intent of finding any errors it may contain. A good test is one with a high probability of finding an as yet undiscovered error; a successful test is the one that uncovers an as yet undiscovered error. Therefore if there are no errors found in testing or, in other words, if the testing is unsuccessful, then it demonstrates that the software functions appears to be working according to specification but this can not be guaranteed for most software, as exhaustive testing is usually impossible. Testing can also demonstrate that the performance of software meets the requirement. Data collected during testing also provides a good indication of software reliability and software quality.

Testing activities can be normally divided into software ***verification*** and software ***validation***. Software verification refers to activities to ensure that the software correctly implements specific functions. Software validation refers to activities to ensure that the software really meets the customer requirements. These two terms can be summarised as two questions (Boehm, 1981):

- Verification: are we building the system correctly?
- Validation: are we building the correct system?

Validation requires the active involvement of customers, the purpose of validation is to make sure that the system is built correctly and is acceptable by the customers. If there are any errors found during this stage, it would be too late to correct them.

In order to assure that the software system can be delivered to customers for validation with high quality, software verification should be interleaved into the software development life cycle. There are many testing techniques which can be used in software development (Pressman, 1997a).

Testing can be carried out in different levels:

- *Unit testing*: it is to test the smallest unit of software – the module, in order to verify its requirements is met.

- *Integration testing*: it is a testing with the purpose of verifying that the units are working together correctly. Integration testing sought to detect inconsistencies between the software modules. Integration testing can be *non-incremental integration* – putting everything together at once and testing as a whole, but this is usually a disaster if the system is big. It can also be *incremental integration* – the program is constructed and tested in small segments, therefore errors are easier to isolate and correct. A number of different incremental integration strategies can be used: *top-down integration*, *bottom-up integration* and *regression testing* (Pressman, 1997b).
- *System testing*: it is to test the entire system or the application, this takes the end-user view of the system. Once all the software modules have been integrated and the integrated software system itself integrated with any hardware subsystems, the time has come to carry out system testing. System testing is the first time at which the entire system can be tested against the system specification.

In software testing, you can never usually prove that the program will never fail. Then one question arises: "How do we know that we have tested enough?" Two responses could be:

1. Never. The testing simple shifts from developers to customers or users.
2. When you run out of time or money.

For the AudioGraph security system, unit testing and integration testing have been interleaved into the system development process, and have been carried out by the author during the development and coding of the system. The testing was carried out intensively until no errors were found and until the author thought it was enough. Unit testing and integration testing of the AudioGraph security system will not be discussed any further.

After all the modules of AudioGraph security system have been integrated and the integration testing fails to find any errors, it is time to carry out system testing. System testing is actually a series of different tests whose primary purpose is to fully exercise the software under test. The goal these tests are to verify that all system elements have been properly integrated and perform the allocated functions. There are different types

of system testing. For instance, *Recovery testing* is to test the fault tolerance of the system; *security testing* is to test the capability of the system to protect sensitive information against from improper or illegal use; *stress testing* is to test the system with abnormal situations like high frequency of inputs; *performance testing* is to test the run-time performance of software within the context of an integrated system. Since the essential purpose of building AudioGraph security system is to protect the AudioGraph tool and courseware material from illegal use, we are only going to verify the security function of AudioGraph security system. Therefore only security testing will be carried out and be demonstrated in the remaining part of this chapter.

6.1 Testing Strategy

6.1.1 System Requirements Specification

As mentioned in chapter four, AudioGraph security system consists of three related parts: the *Copy Protection Record* in .aep file, the *Key Insertion Tool* and *Usage Control embedded into AudioGraph Plug-in* (We would refer to the AudioGraph Plug-in with Usage Control as *Secure AudioGraph Plug-in* in this chapter). In order to verify the AudioGraph security system, we now describe system requirement specification as following:

- The Key Insertion Tool should allow the user to specify or select an .aep file.
- The Key Insertion Tool should be able to detect the existence or absence of the Copy Protection Record in the specified .aep file.
- The Key Insertion Tool should be able to extract a Copy Protection Record inside the specified .aep file if it does contain a Copy Protection Record.
- The Key Insertion Tool should be able to insert a Copy Protection Record into the specified .aep file if it does not contain a Copy Protection Record.
- The Key Insertion Tool should be able to modify the contents (the *Company Key*, the *Lower protection key*, and the *Upper protection key*) of the Copy Protection Record in the specified .aep file if it does contain a Copy Protection Record.
- Whenever a user specifies a userID and if there is a Copy Protection Record, the Key Insertion Tool should be able to determine the unique password for this specified userID.

- The Web browser with the Secure AudioGraph Plug-in should be able to playback a presentation in an .aep file that does not contain a Copy Protection Record.
- On playing back a presentation in an .aep file that does contain a Copy Protection Record, the Secure AudioGraph Plug-in should force users to specify a userID/password pair.
- On playing back a presentation in an .aep file that does contain a Copy Protection Record, if users fail to specify a correct userID/password pair, the Secure AudioGraph Plug-in should disable the play back of the presentation.
- On playing back the presentation in an .aep file that does contain a Copy Protection Record, if users succeed in specifying a correct userID/password pair, the Secure AudioGraph Plug-in should playback the presentation.
- On playing back the presentation in an .aep file that does contain a Copy Protection Record, if users have ever specified a correct userID/password pair in previous presentations, and the previous .aep files contain the same Copy Protection Record as current .aep file, then the Secure AudioGraph Plug-in should playback the presentation, without asking user to specify userID/password pair again.
- The password should never appear in plaintext format when users specify the userID/password pair.
- The password should never be stored on local disk in plaintext format.

With above system requirement specification, we can now work out testing plan to verify that the system requirements have been met.

6.1.2 Test Plan

The system testing takes .aep files as input data, these files do not contain the Copy Protection Record. Then it starts to very the function of the Key Insertion Tool for detecting, inserting, extracting or updating of Copy Protection Record into the specified .aep file. Then it starts to verify the function of password generation in Key Insertion Tool for the specified userID. The .aep file and the userID/password pair will be used as input to the testing of Secure AudioGraph Plug-in.

The Usage Control testing of Secure AudioGraph Plug-in is to verify its Usage Control function. The testing will verify that only after user specifying a correct userID/password pair, the presentation can be played back. Without a correct userID/password pair, there is not way to bypass this Usage Control and playing back the presentation.

Test Plan for Key Insertion Tool

1. Launch Key Insertion Tool.
2. Select the file `E:\AudioGraph\Security Test\Regre028.aep` into Key Insertion Tool.
3. Detect whether the file contains a Copy Protection Record. Verifying that Key Insertion Tool can not detect the Copy Protection Record in the file.
4. Specify a Copy Protection Record (the *Company Key*, the *Lower protection key*, and the *Upper protection key*), then insert this Copy Protection Record into the specified .aep file. The Copy Protection Record is:

   Company Key = *1111122222*
   Lower Protection Key = *1111111111*
   Upper Protection Key = *1000000000*
5. Extract Copy Protection Record from `E:\AudioGraph\Security Test\Regre028.aep`. Verify that the Copy Protection Record is correct (it should be the same as is in step 5).
6. Change the Copy Protection Record to following value:

   Company Key = *2147483647*
   Lower Protection Key = *1111111111*
   Upper Protection Key = *1000000000*
7. With above Copy Protection Record, specifying a userID, said *userID = " myUserID"*, then the Key Insertion Tool will automatically generate a password for users, i.e., *password = "B9QoFukhG8i"*.
8. Using same method as in step 3 to step 5, to insert the same Copy Protection Record into another .aep file, said file: `E:\AudioGraph\Security Test\Regre029.aep`

The output .aep files here and the userID/password will be used in the testing plan for Secure AudioGraph Plug-in.

Test plan for Secure AudioGraph Plug-in

9. Playing back a presentation, whose .aep file does not contain a Copy Protection Recorder, verifying that it does not require user to specify userID/password. The .aep file used here is "`E:\AudioGraph\Security test\Regre027.aep`".
10. Playing back presentation in .aep file: "`E:\AudioGraph\Security test\Regre028.aep`". Since it contains a Copy Protection Record, we need to verify that it will force users to specify a userID/password, and verify that users can not bypass the userID/password to playback the presentation.
11. In order to test its Usage Control function, we try some combination of userID/password as showed below, among them only the last userID/password pair is a correct userID/password pair. Verifying that only the correct userID/password can pass the Usage Control test to playback the presentation, other userID/password can not pass the Usage Control test and will lead to disable the playing back of presentation.

    userID = "nyUserID", password = "B9QoFukhG8i"
    userID = "lyUserID", password = "B9QoFukhG8i"
    userID = " myuserID", password = "B9QoFukhG8i"
    userID = " myUserJD", password = "B9QoFukhG8i"
    userID = " myUserIC", password = "B9QoFukhG8i"
    userID = "myUserID", password = "B8QoFukhG8i"
    userID = " myUserID", password = "B9QoFUkhG8i"
    userID = " myUserID", password = "B9QoFukhG7i"
    userID = " myUserID", password = "B9QoFukhG9i"
    userID = " myUserID", password = "B9QoFukhG8i"

12. Before we input the correct userID/password pair, we can test as more other userID/password combination as we like, in order to further verify its Usage Control function.
13. After we have specified the correct userID/password pair, we then try to playback the presentation in file `E:\AudioGraph\Security test\Regre029.aep`, which contains the same Copy Protection Record as file

`E:\AudioGraph\Security test\Regre028.aep`. Verifying that it does not require users to specify the userID/password again.

14. Open file for storing userID/password, i.e., file "`C:\ AepProduction.dat`", with any text editor software, we uses Microsoft Notepad here, to verify that the content of this file is unreadable (not in plaintext format).

6.1.3 System Configuration

The following system configuration provide us with the environment for the practical testing, so they should be performed before we start testing:

- Computer System: Intel Celeron 533MHz, 492Mb RAM, 50G HDD
- Operating System: Windows 2000 SP2
- Web Browser: Netscape Communicator 4.79
- Key Insertion Tool: executable file "`E:\AudioGraph\Security Test\SetKey.exe`"
- Secure AudioGraph Plug-in: file "`Npaep.dll`" installed in folder "`C:\Program Files\Netscape\Communicator\Program\Plugins`"
- Dynamic Link Library files required by Secure AudioGraph Plug-in: `gsm.dll`, `png.dll` and `zlib.dll`. They are all located in folder "`C:\Program Files\Netscape\Communicator\Program\Plugins`"
- Test files location: all test files located in folder "`E:\AudioGraph\Security Test\`". Test files include:

```
Regre027.htm, Regre027.aep
Regre028.htm, Regre028.aep
Regre029.htm, Regre029.aep
```

File "`Regre027.htm`" contains a tag to embed the file "`Regre027.aep`":

```
<EMBED SRC="Regre027.aep" HEIGHT="347" WIDTH="184"
    ALIGN="BOTTOM">
```

File "`Regre028.htm`" contains a tag to embed the file "`Regre028.aep`":

```
<EMBED SRC="Regre028.aep" HEIGHT="371" WIDTH="504"
    ALIGN="BOTTOM">
```

File "`Regre029.htm`" contains a tag to embed the file "`Regre028.aep`":

```
<EMBED SRC="Regre029.aep" HEIGHT="321" WIDTH="454"
    ALIGN="BOTTOM">
```

In the following sections we are going to demonstrate some of the system testing according to the above test plan. The following sections also demonstrate some of the testing results.

6.2 Testing of Key Insertion Tool

After the Key Insertion Tool has been launched, the main interface of Key Insertion Tool appears, as illustrated in Figure 6.1:

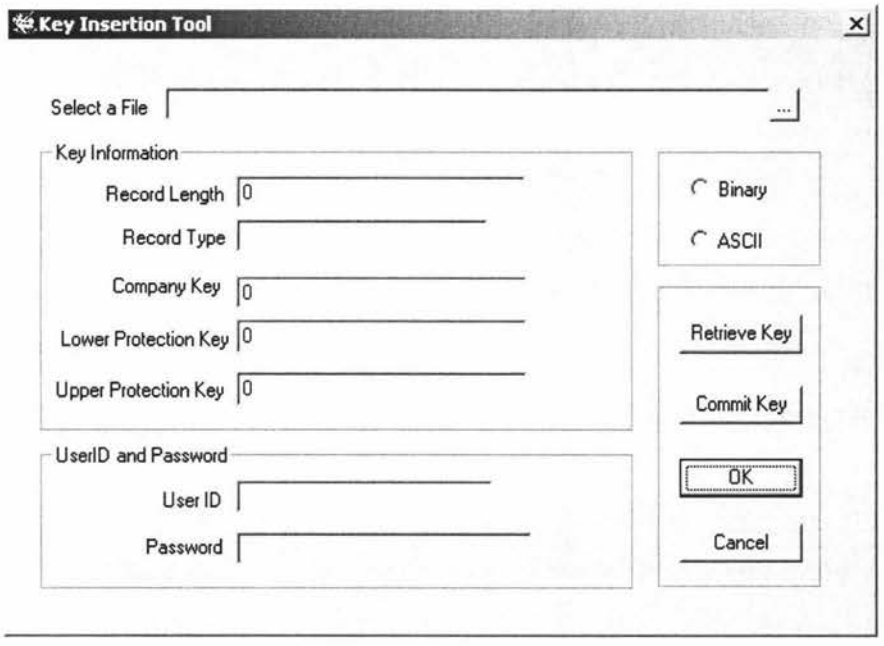


Figure 6.1: Main Interface of Key Insertion Tool

Now we can select an .aep file by click on the button besides the filename field. After we choose file:

```
E:\AudioGraph\Security Test\Regre028.aep
```

It starts to detect whether it contains a *Copy Protection Record*. As this file does not contain a *Copy Protection Record*, it display a message window showing that it does not contain a Copy Protection Record, as illustrated in Figure 6.2:

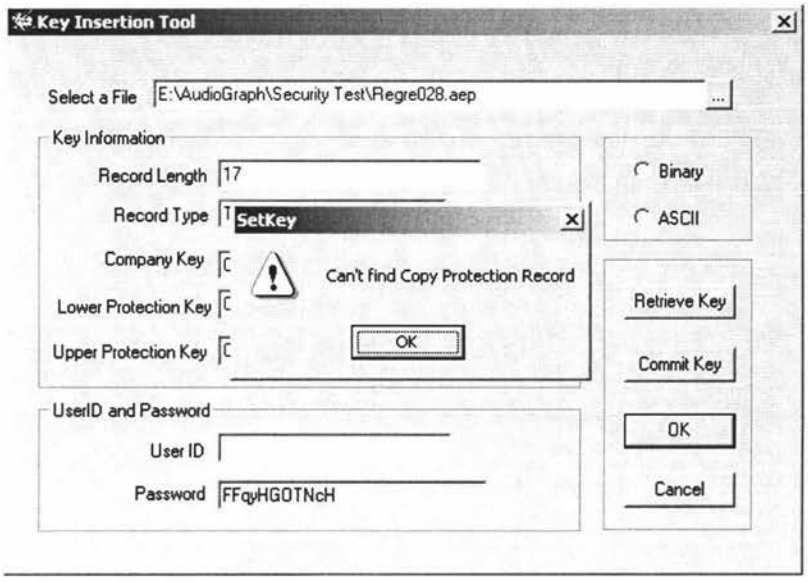


Figure 6.2: Can not Find Copy Protection Record

Then we can specify a Copy Protection Record with following values:

Company Key = *1111122222*

Lower Protection Key = *1111111111*

Upper Protection Key = *1000000000*

as is illustrated in Figure 6.3. After we click on the button "Commit Key", a Copy Protection Record will be inserted into the specified file "*E:\AudioGraph\Security Test\Regre028.aep*".

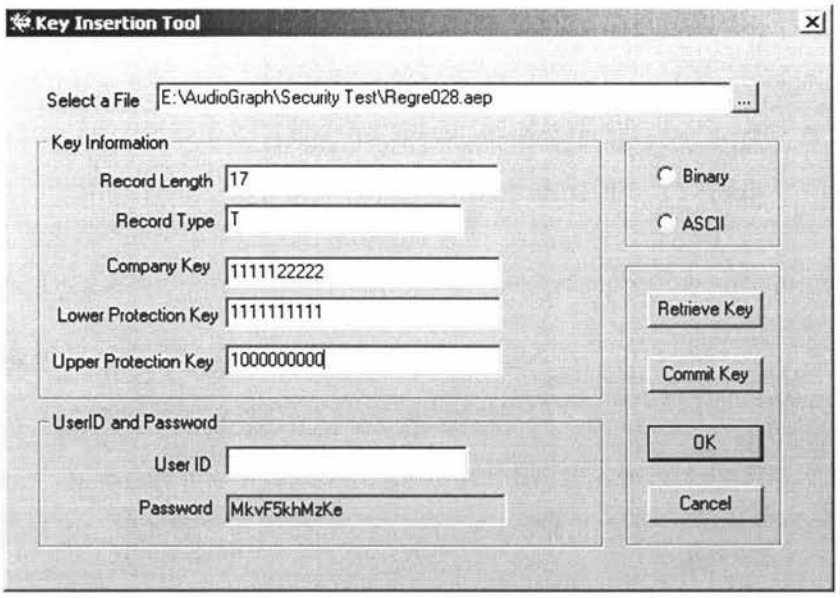


Figure 6.3: Specifying a Copy Protection Record

Then we can verify that the Copy Protection Record is really inserted into the .aep file, by means of retrieving the record from the file again. As the file name is already been specified, we can just click on button "Retrieve Key" to extract the Copy Protection Record from the file. The result is illustrated in Figure 6.4. After click on the "OK" button, we can check the value of the Copy Protection Record and verify that its value is the same as that in Figure 6.3.

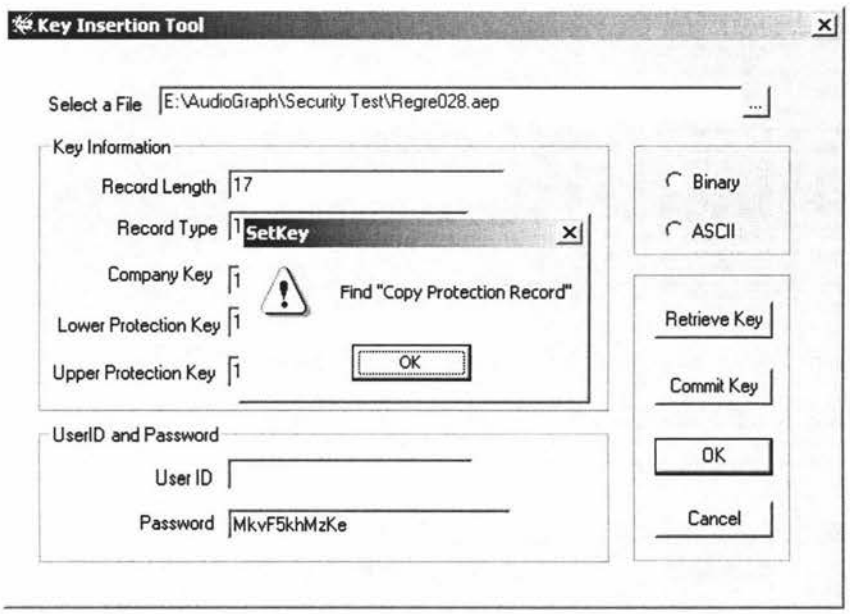


Figure 6.4: Verifying Copy Protection Record

Then we can modify the Copy Protection Record to new value as specified below:

Company Key = *2147483647*

Lower Protection Key = *1111111111*

Upper Protection Key = *1000000000*

After we click on button "Commit Key", the Copy Protection Record in the specified .aep file will be updated. We can click on button "Retrieve Key" again to read the Copy Protection Record in .aep file and verify its correctness.

Now we are going to generate password for a specified suerID. With Copy Protection Record as:

Company Key = *2147483647*

Lower Protection Key = *1111111111*

Upper Protection Key = *1000000000*

after we specify a userID, i.e., *userID =" myUserID"*, the Key Insertion Tool will automatically generate the corresponding password for us, e.g., *password = "B9QoFukhG8i"*. This is demonstrated in Figure 6.5.

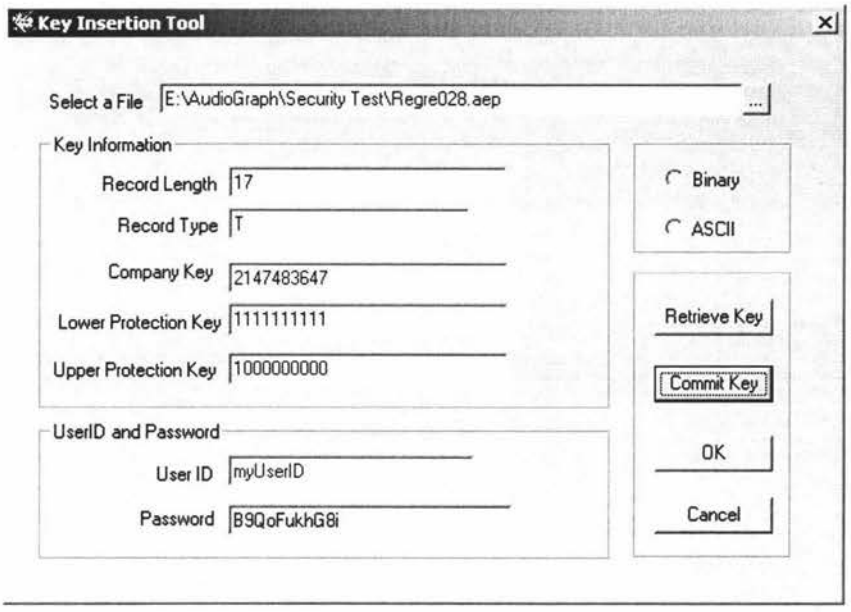


Figure 6.5: Generating a Password for Specified userID

We have inserted and updated a Copy Protection into file `E:\AudioGraph\Security Test\Regre028.aep`. By using the same method, we can insert the same Copy Protection into file `E:\AudioGraph\Security Test\Regre029.aep`. We are not going to demonstrate this again. These two .aep files will be used in following testing.

6.3 Testing of Secure AudioGraph Plug-in

First we are going to playback presentation whose .aep file does not contain a Copy Protection Record. Since we have never inserted Copy Protection Record into file `E:\AudioGraph\Security Test\Regre027.aep`, it does not contain the Copy Protection Record. As soon as we open its html file, i.e., file `E:\AudioGraph\Security Test\Regre027.htm` from Netscape Communicator, it start to playback the presentation, without asking user to specify a userID/password, as illustrated in Figure 6.6.

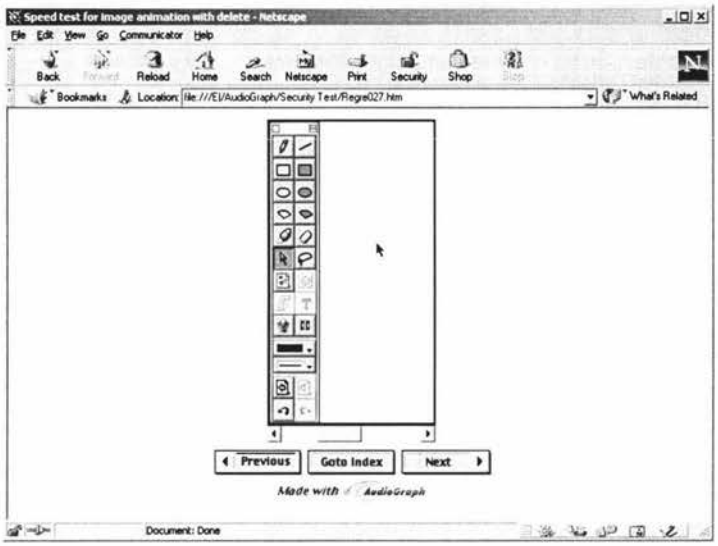


Figure 6.6: Playback of Presentation Which Does Not Contains the Copy Protection Record

Then we start to playback presentation that contains a Copy Protection Record. After Netscape Communicator opens the html file `E:\AudioGraph\Security Test\Regre028.htm`, it starts to playback presentation in file `E:\AudioGraph\Security Test\Regre028.aep`. Since we have ever inserted a Copy Protection Record into this .aep file, it does contain a Copy Protection Record. As we expected, it force user to input a userID/password pair, as illustrated in Figure 6.7:

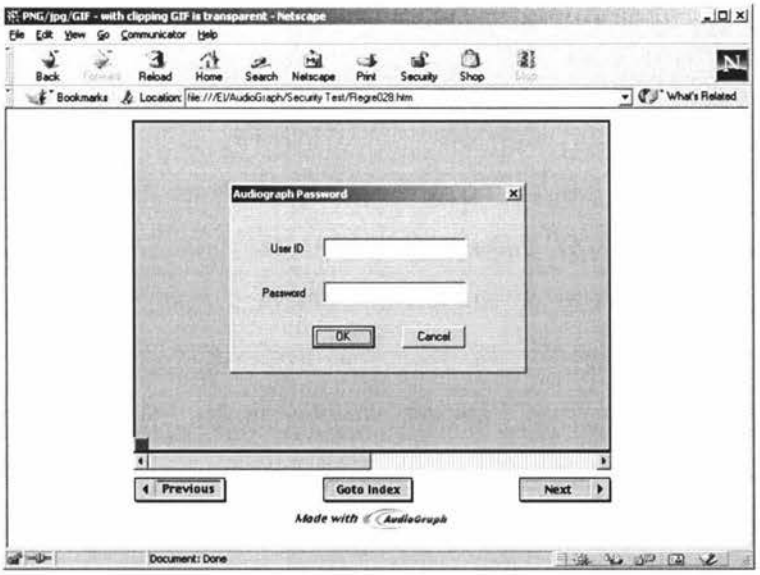


Figure 6.7: Playback of Presentation Which Does Contains a Copy Protection Record

Then we can input userID/password here. We try the following userID/password combinations:

userID = "nyUserID", password = "B9QoFukhG8i"
userID = "lyUserID", password = "B9QoFukhG8i"
userID = " myuserID", password = "B9QoFukhG8i"
userID = " myUserJD", password = "B9QoFukhG8i"
userID = " myUserIC", password = "B9QoFukhG8i"
userID = "myUserID", password = "B8QoFukhG8i"
userID = " myUserID", password = "B9QoFUkhG8i"
userID = " myUserID", password = "B9QoFukhG7i"
userID = " myUserID", password = "B9QoFukhG9i"
userID = " myUserID", password = "B9QoFukhG8i"

The testing result is demonstrated in Table 6.1. Refer back to Figure 6.5 in previous section, we found that only the correct userID/password (the last pair) can pass the Usage Control test.

Table 6.1: userID/password Testing Result

userID	***password***	***Pass the Usage Control Test?***
nyUserID	*B9QoFukhG8i*	No
lyUserID	*B9QoFukhG8i*	No
myuserID	*B9QoFukhG8i*	No
myUserJD	*B9QoFukhG8i*	No
myUserIC	*B9QoFukhG8i*	No
myUserID	*B8QoFukhG8i*	No
myUserID	*B9QoFUkhG8i*	No
myUserID	*B9QoFukhG7i*	No
myUserID	*B9QoFukhG9i*	No
myUserID	*B9QoFukhG8i*	Yes

If the user inputs an incorrect userID/password pair, the Plug-in will disable the play back of the presentation, and a warning message is displayed to inform the user, as illustrated in Figure 6.8. Every time when the users tries to playback this presentation, the window illustrated in Figure 6.8 will always appear.

If users input a correct userID/password pair, then it will pass the Usage Control test and the presentation will be played back normally, as illustrated in Figure 6.9.

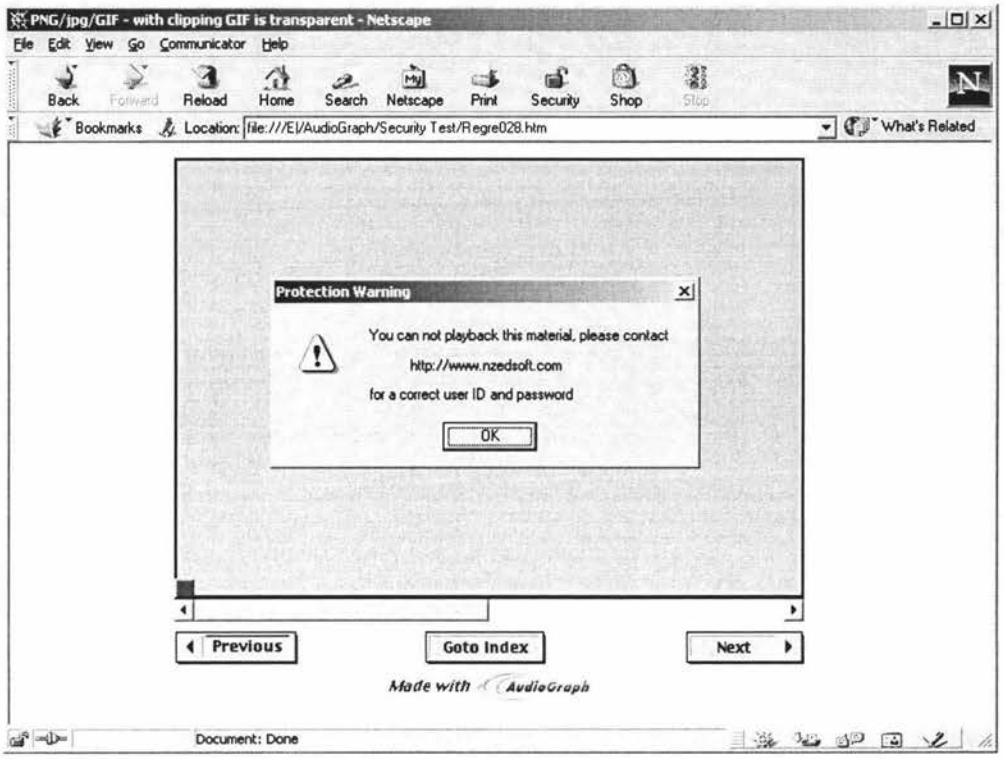


Figure 6.8: Refuse to Play Back When Input Incorrect userID/password

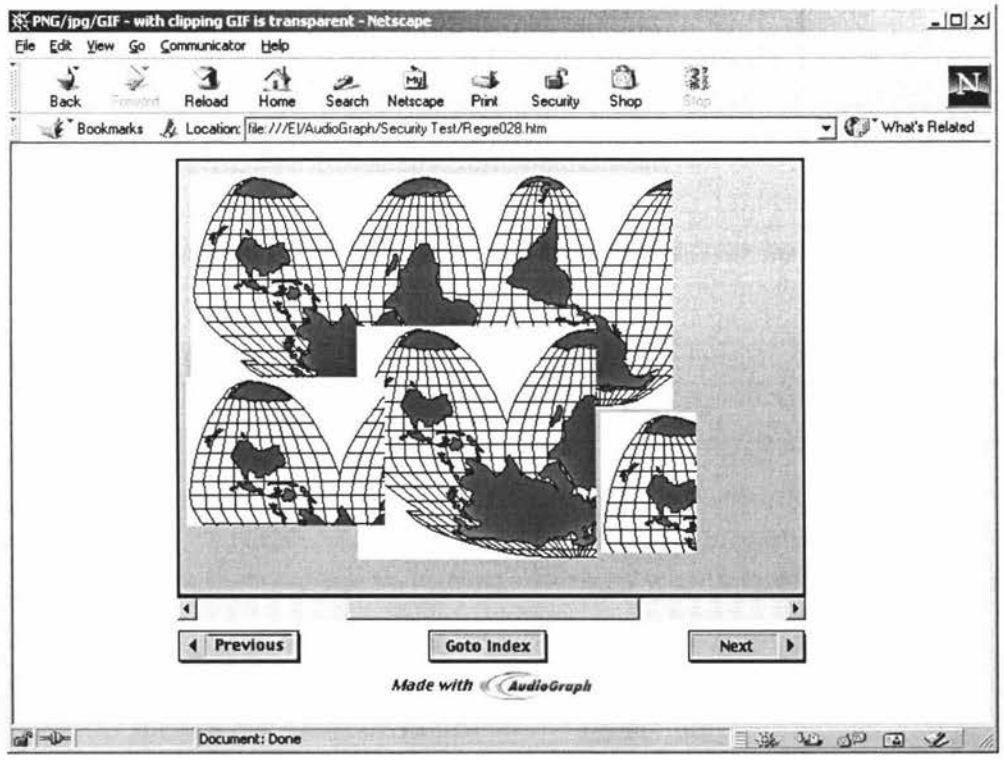


Figure 6.9: Playback Presentation If Input Correct userID/password Pair

Now that we have already input a correct userID/password pair, the next time the plug-in plays back a presentation, which contains the same Copy Protection Record in its .aep file, it would not require the user to input the userID/password again. We can verify this by opening file `E:\AudioGraph\Security Test\Regre029.htm` in Netscape Communicator. As soon as we open this html file, it starts to playback the presentation in .aep file `E:\AudioGraph\Security Test\Regre029.aep`, as illustrated in Figure 6.10. Because the Copy Protection Record in this file is the same as in previous file `E:\AudioGraph\Security Test\Regre028.aep`, it will not ask us to specify the userID/password again.

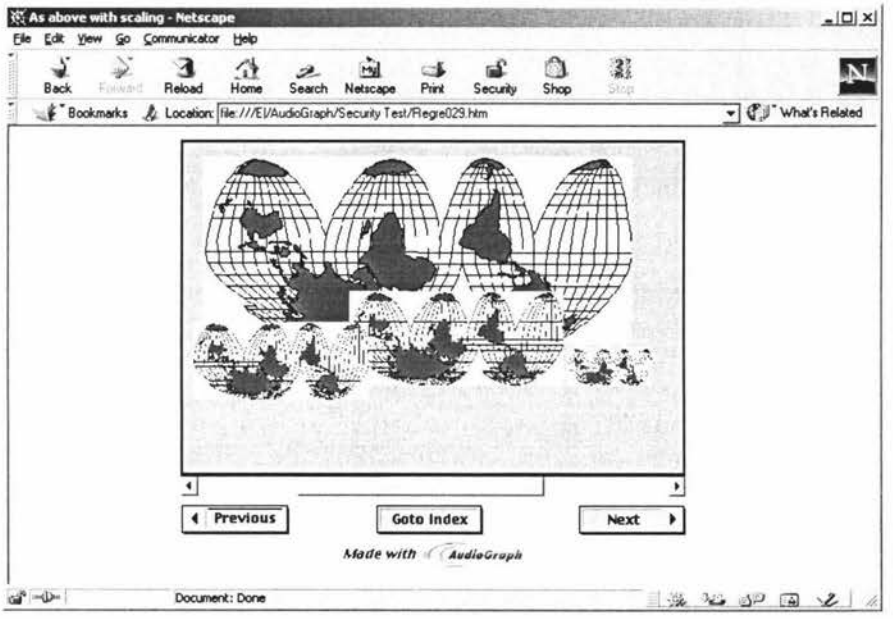


Figure 6.10: Playback Presentation Which Contains the Same Copy Protection as Previous Presentation

The last thing we want to demonstrate is that the password will never be stored on local disk in plaintext format. After we have ever input a correct userID/password, this userID/password will be stored in the local disk file "`C:\AepProduction.dat`". Now let us open this file by Microsoft Notepad, the displayed message is unreadable, as illustrated in Figure 6.11. This shows that the userID/password is not in plaintext format.

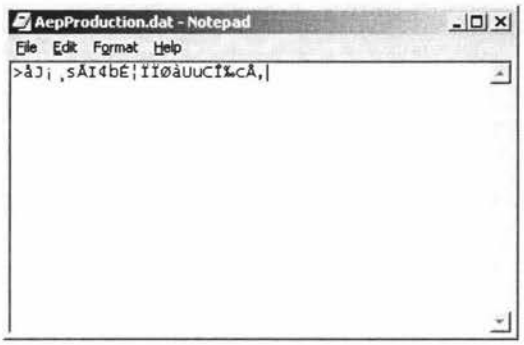


Figure 6.11: Content of File To Store UserID/Password File

6.4 Conclusion

This chapter has introduced the concept of software testing in software engineering. Then the system requirement specification of the AudioGraph security system was described in detail. In order to verify that AudioGraph security system meets the system requirement, a testing plan was worked out and practical testing was carried out according to the test plan. The results of testing demonstrate that no errors are found during system testing.

On testing of Key Insertion Tool, we have verify that the Key Insertion Tool can detect, insert, extract and update the Copy Protection Record in a specified .aep file correctly. The functionality of the Key Insertion Tool also appears to be working to specifications.

On the Secure AudioGraph Plug-in testing, we have only tested a few of the userID/password combinations, they all demonstrate that the Usage Control function inside Secure AudioGraph Plug-in does not fail. However this is not enough to prove that it will never fail. Since the number of userID/password combination is enormous, it is impractical to test all the userID/password combinations. With the limit of time, we can only test a small number of the userID/password combination, the author consider these tests would be enough to demonstrate the function of Usage Control inside AudioGraph Plug-in. Given more time and resource, more of the userID/password can be tested, and that would definitely improve the quality of system testing.

Chapter 7: Conclusions

7.1 Summary of the Thesis

This thesis has covered the various areas related to the design and implementation of AudioGraph security system. The AudioGraph project and the technologies used in AudioGraph were described briefly in the first chapter. The first chapter also introduced the goal and requirements of the AudioGraph security system and introduced some concepts related to the computer security. This was intended to give the reader a feel for the overall subject area.

Cryptography plays a center role in Information Security. It is critical to choose a good and suitable cryptography algorithm when design a security system. To choose a cryptography algorithm requires to understand the cryptography to some extent. Chapter two overview some of the most important modern cryptography algorithms. The modern cryptography algorithms can be classified into two categories: conventional and public-key encryption algorithms. In conventional encryption algorithms, two of the standard encryption algorithms – DES and AES were introduced. IDEA is also a popular used conventional algorithm because it is used in PGP system, so it was described in chapter two as well. In public-key encryption algorithm, two algorithms were introduced – Knapsack algorithm and RSA algorithm; Knapsack is the first public-key algorithm, but it has been broken. RSA algorithm is nearly the only popular used public-key algorithm.

Another important area in information security is key management. Usually the cryptography algorithms are in the public domain and are known to the world, only the key of the encryption algorithm should be kept secret. Therefore key management is also quite important in information security, the issues related to key management are discussed in Chapter three. Chapter three also introduced concepts referred to as *watermarking* and *fingerprinting* – which are digital copyright labeling techniques used to protect electronic digital documents.

From chapter four it starts to discuss the AudioGraph security system. Chapter four gave two examples which used encryption algorithm to archive security. One is password protection system in the UNIX operating system, which used variant of DES algorithm to encrypt the contents of the password in related password file. Another example is the SSL/TLS architecture. In SSL/TLS, a secure Web server required to get a X.509 Certificate of its public/private key pair, then in procedure of SSL/TLS handshaking, a Web server and a Web client (i.e., Web Browser) require to exchange session key using certified public key. After giving these two example, chapter four described a proposed security system which meets the security requirements of AudioGraph project, this security system consists of three parts: a *Copy Protection Record* in the .aep file; a Key Insertion Tool to detect, extract, update and insert a Copy Protection record and to determine the userID/password pair; and a scheme of Usage Control inside the AudioGraph player (i.e., AudioGraph Plug-in, etc.). The security system of AudioGraph was described in chapter four in detail. It discussed the scheme of Usage Control scheme embedded in AudioGraph Plug-in and the selection of encryption algorithms. The security of this system had been discussed, it had been proved that this system is strong enough to against brute-force attack and other known cryptanalysis.

The implementation of the security system and encryption algorithms had been described in chapter five, the algorithm and program of Blowfish and TEA had been given, the method to convert arbitrary bits stream into printable character string was described. Chapter six gave testing result of the system, including result of testing on the Key Insertion Tool and testing on the Usage Control inside the AudioGraph Plug-in. The results demonstrated that the goal of AudioGraph security system had been achieved by the implementation of the system.

In summary, this thesis described the design, implementation and system testing of a security system to protect AudioGraph courseware material from being illegal use. In the thesis, the security system had also been proved theoretically to be very strong against all known cryptanalysis and other attacks. The results from system testing proved that the security system succeed in archiving its goal.

7.2 Future Work

The Key Insertion Tool has already been implemented, the security system has already been implemented in AudioGraph Plug-in for Microsoft Windows. This ensures that the AudioGraph presentation material can be protected by our security system.

The AudioGraph Tools consist Recorder for Windows, Recorder for Macintosh, Player for Macintosh, Plug-in for Macintosh and Plug-in for Windows. At the moment, the Protection system has only been implemented in AudioGraph Plug-in for Windows, the protection system requires to be implemented in other AudioGraph Player, i.e., AudioGraph Player for Macintosh and AudioGraph Plug-in for Macintosh as well.

As has been discussed in Chapter three, key management has critical importance in computer security. For our AudioGraph, it is necessary to well manage the generating, the distribution, the storage, the updating of userID/password pair. In the current AudioGraph security system design and implementation, there are no policy or tool for AudioGraph useID/password management. Further research, design and implementation of AudioGraph key management system is required to guarantee the security of the whole system. According to discussion in Chapter three, a randomly chosen userID is better than a meaningful userID, therefore the key management system should be designed to generate a random userID for user to choose. The key management system should also consider the distribution of userID/password to the AudioGraph material consumer in secure channel. There must be a database for storage of userID/password pairs in the AudioGraph key management system, this database should also be a secure database, i.e., only the authority person can access to the userID/password database.

As has been discussed in Chapter four, the encryption algorithms used in AudioGraph security system are Blowfish and TEA algorithms. They are still unbreakable. However, there is always possibility for cryptanalysis to make breakthrough. So it is always necessary to keep eyes on the breakthrough of cryptanalysis, especially the cryptanalysis of Blowfish and TEA algorithm. In case the Blowfish or TEA had been broken, we should change the algorithms, and that would be a big change for the AudioGraph security system, and that would always have big impact on the whole system.

Fortunately, up to now there are no many progress on cryptanalysis of Blowfish and TEA, they are still very strong algorithms and are increasingly accepted by the computer security community.

References

Adams, C. (1997). RFC2144 – The CAST-128 Encryption Algorithm. [Online] Available: http://www.rfc.net/rfc2144.html (Retrieved May 6, 2000)

Anderson, R. (2001). SERPENT – A Candidate Block Cipher for the Advanced Encryption Standard. [Online] Available: http://www.cl.cam.ac.uk/~rja14/serpent.html (Retrieved October 11, 2001)

Atkins, D. (1991). Request for Comments: 1991 - PGP Message Exchange Formats. [Online] Available: http://www.wu-wien.ac.at:8082/rfc/rfc1991.hyx/$$root (Retrieved April 23, 2002).

Beresford, R. (2002). Finite-Field Arithmetic. [Online] Available: http://www.anujseth.com/crypto/ffield.html. (Retrieved May 11, 2002)

Biham, E. (1994). New Types of Cryptanalytic Attacks Using Related Keys. *Journal of Cryptology, v.7, n.4, 1994*, 229-246.

Biham, E. and Shamir, A. (1990). Differential Cryptanalysis of DES-like Cryptosystems. *Advances in Cryptology-CRYPTO '90 Proceedings,* Springer-Verlag, *1991*, 2 – 21.

Biham, E. and Shamir, A. (1992). Differential Cryptanalysis of the Full 16-Round DES. *Advances in Cryptology-CRYPTO '92 Proceedings*, Springer-Verlag, *1993*, 487 – 496.

Boehm, B. (1981). *Software Engineering Economics*. New Jersey, U.S.A.: Prentice-Hall Inc. pp. 37

Boeyen, S., Howes, T., & Richard, P. (1999). Request for Comments: 2559 - Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2. [Online] Available: http://www.ietf.org/rfc/rfc2559.txt (Retrieved December 15, 2001)

Boneh, D., & Shaw, J. (1995). Collusion-Secure Fingerprinting for Digital Data. *Advances in Cryptology – CRYPTO'95*, Springer-Verlag, 452 – 465.

Borenstein, N., & Freed, N. (1992). RFC 1341 – MIME (Multipurpose Internet Mail Extensions). [Online] Available: http://sunsite.dk/RFC/rfc/rfc1341.html (Retrieved May 6, 2000)

Borenstein, N., & Freed, N. (1993). RFC 1521 – MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. [Online] Available: http://www.rfc.net/rfc1521.html (Retrieved May 6, 2000)

Boutell, T., & Lane, T. (Ed.). (1996). PNG (Portable Network Graphics) Specification Version 1.0. [Online] Available: http://www.w3.org/TR/png.html (Retrieved October 02, 2001)

Browne, S. (1997). X.509 Certificates and Certification Authorities. [Online] Available: http://www.cs.utk.edu/~browne/nhse-legal/node12.html. (Retrieved December 15, 2001)

Brown, L. (1998). A Current Perspective on Encryption Algorithms. [Online] Available: http://www.adfa.oz.au/~lpb/papers/cauugs98/ (Retrieved December 12, 1999)

Cao, Z.F., & Zhao, G. (1994). Some New MC Knapsack Cryptosystems. *CHINACRYPT'94, xidian, China, 11-15 Nov 1994*, 70 – 75.

Chor, B., & Rivest, R.L. (1984). A Knapsack Type Public Key Cryptosystem Based on Arithmetic in Finite Fields. *Advances in Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag, 54 – 65.

Counterpane Internet Security Inc. (1999a). Speed Comparisons of Block Ciphers on a Pentium. [Online] Available: http://www.counterpane.com/speed.html (Retrieved September 14, 1999).

Counterpane Internet Security Inc. (1999b). Twofish - Performance vs. Other Block Ciphers (on a Pentium). Available: http://www.counterpane.com/twofish_performance.html (Retrieved September 30, 1999).

Counterpane Internet Security Inc. (2000). The Blowfish Encryption Algorithm. [Online] Available: http://www.counterpane.com/blowfish.html (Retrieved May 6, 2000)

Counterpane Internet Security Inc. (2001). Twofish: A New Block Cipher. [Online] Available: http://www.counterpane.com/twofish.html (Retrieved October 11, 2001)

Counterpane Internet Security Inc. (2002). Products that use Blowfish. [Online] Available: http://www.counterpane.com/products.html (Retrieved January 12, 2002).

Daemen, J., Knudsen, L.R., Rijmen, V. (1997). The Block Cipher Square Algorithm. [Online] Available: http://www.ddj.com/documents/s=936/ddj9710e/9710e.htm (Retrieved October 11, 2001)

Daemen, J. & Rijmen, V. (1999). AES Proposal: Rijndael. [Online] Available: http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf (Retrieved October 18, 2001)

Degener, J. (2000). GSM 06.10 lossy speech compression. [Online] Available: http://kbs.cs.tu-berlin.de/~jutta/toast.html (Retrieved October 14, 2001)

Degener, J., & Bormann, C. (2000a). Digital Speech Compression - Putting the GSM 06.10 RPE-LTP algorithm to work. [Online] Available: http://www.ddj.com/articles/1994/9412/9412b/9412b.htm (Retrieved October 15,

2001)

Degener, J., & Bormann, C. (2000b). gsm-1.0.10.tar.gz. [File archive online] Available: ftp://ftp.cs.tu-berlin.de/pub/local/kbs/tubmik/gsm/gsm-1.0.10.tar.gz (Retrieved October 15, 2000)

Degener, J., & Bormann, C. (2000c). GSMDecoder-1.0-src.tar.gz. [File archive online] Available: http://pix.test.at/gsm/GSMDecoder-1.0-src.tar.gz (Retrieved October 15, 2000)

Deutsch, P. (1996). RFC 1950 – ZLIB Compressed Data Format Specification version 3.3. [Online] Available: http://www.rfc.net/rfc1950.html (Retrieved October 03, 2000)

Dierks, T., & Allen, C. (1999). RFC 2246 – The TLS Protocol Version 1.0, [Online] Available: http://www.ietf.org/rfc/rfc2246.txt (Retrieved March 25, 1999)

Diffie, W. & Hellman, M.E. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory. v.IT-22, n.6, Nov 1976*, 644 – 654.

Emeterio, A. S. (1999). LZ77 the basics of compression (2nd ed.). [Online] Available: http://www.arturocampos.com/ac_lz77.html (Retrieved October 1, 2001)

Fridrich, J. (1998). Robust Digital Watermarking Based on Key-Dependent Basis Functions. *Information Hiding, 1998*, Springer-Verlag, 143 – 157

Freier, A.O., Karlton, P., & Kocher, P.C. (1996). The SSL Protocol Version 3.0 Internet Draft. [Online] Available: http://home.netscape.com/eng/ssl3/ssl-toc.html (Retrieved August 11, 1999)

Garfinkel, S. (1997). *Web Security & Commerce*. U.S.A: O'Reilly & Associates Inc.

Garfinkel, S., & Spafford, G. (1991). *Practical UNIX Security*, O'Reilly & Associates Inc., 1991.

HART, G., & MASON, J. (1996). *The virtual university*. Symposium proceedings, 1996, The University of Melbourne, ISBN 0 73251 428 2

Hellman, M.E. (1976). DES will be Totally Insecure within Ten Years. *IEEE Spectrum, v.16, n.7, Jul 1979*, 32-39.

Hellman, M.E.(1979). The Mathematics of Public-Key Cryptography. *Scientific American, V.241, n.8, Aug 1979*, 146 – 157

Herrige, A., Ruanaidh, J., Petersen, H., & Pereira, S. (1998). Secure Copyright Protection Techniques for Digital Images. *Information Hiding, 1998*, Springer-Verlag, 169 – 189

Hussain, H.A., Sada, J.W.A., & Kalipha, S.M. (1991). New Multistage Knapsack Public-Key Cryptosystem. *International Journal of Systems Science, V.22, n.11, Nov 1991*, 2313 – 2320

Huth, M. (2001). CIS 798. Secret-sharing protocol. [Online] Available: http://www.cis.ksu.edu/~huth/cry/secret.html (Retrieved December 14, 2001)

IBM Research (2001). The MARS cipher – IBM submission to AES. [Online] Available: http://www.research.ibm.com/security/mars.html (Retrieved February 5, 2001)

Internet Assigned Numbers Authority (2001). PORT NUMBERS. [Online] Available: http://www.iana.org/assignments/port-numbers (Retrieved December 19, 2001).

Jesshope, C. R., Shafarenko, A., & Slusanschi, H. (1998). Low-bandwidth multimedia tools for Web-based lecture publishing. *IEEE Engineering Science and Educational Journal, 9(4)*, 156-162.

Joancomarti, J.H. (2001). Copyright Protection Schemes for Multimedia Contents. [Online] Available: http://antic.cesca.es/formacio/congressos/tac2000/jherrera-cesca.pdf (Retrieved December 17, 2001)

Kelsey, J., Schneier, B., & Wagner, D. (1997). Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. *ICICS '97 Proceedings*, Springer-Verlag, November 1997, 233-246.

Knudsen, L.R. (1994). *Block Ciphers-Analysis, Design, Application*. Ph.D. dissertation, Aarhus University, Nov 1994.

Kohnfelder, L. (1978). Towards a Practical Public-Key Cryptosystem. Bachelor's Thesis, M.I.T., May 1978.

Lacy, J., Quackenbush, S., Reibman, A., & Snyder, J. (1998). Intellectual Property Systems and Digital Watermarking. *Information Hiding, 1998*, Springer-Verlag, 158 – 168

Langford, S. K. & Hellman, M. E. (1994). Differential-Linear Cryptanalysis. *Advances in Cryptology-CRYPTO '94 Proceedings*, Springer-Verlag, *1994*, 17 – 25.

Lenstra, A. (2001). RSA Factoring-by-Web: General Information. [Online] Available: http://www.npac.syr.edu/factoring/overview.html (Retrieved December 13, 2001)

Linn, J. (1989). RFC 1115 – Privacy Enhancement for Internet Electronic Mail: Part III -- Algorithms, Modes, and Identifiers. [Online] Available: http://www.faqs.org/rfcs/rfc1115.html (Retrieved May 21, 2002).

Massey, J.L., and Lai, X. (1991). *Device for Converting a Digital Block and the Use Thereof.* International Patent PCT/CH91/00117, 28 Nov 1991

Massey, J.L., and Lai, X. (1993). *Device for the Conversion of a Digital Block and Use*

of Same. U.S. Patent #5,214,703, May 25, 1993.

Matsui, M. (1993). Linear Cryptanalysis Method for DES Cipher. *Advances in Cryptology--EUROCRYPT '93 Proceedings*, Springer-Verlag, *1994*, 386 – 397.

Merkle , R.C., & Hellman, M. (1978). Hiding Information and Signature in Trapdoor Knapsacks. *IEEE Transactions on Information Theory, V.24, n.5, Sep 1978*, 525 – 530.

National Institute of Standards and Technology (1993). FIPS PUB 46-2 – Data Encryption Standard (DES). [Online] Available: http://www.itl.nist.gov/fipspubs/fip46-2.htm (Retrieved May 5, 2000)

National Institute of Standards and Technology (1997a). ANNOUNCING DEVELOPMENT OF A FEDERAL INFORMATION PROCESSING STANDARD FOR ADVANCED ENCRYPTION STANDARD. [Online] Available: http://csrc.nist.gov/encryption/aes/pre-round1/aes_9701.txt (Retrieved September 21, 1999)

National Institute of Standards and Technology (1997b). ANNOUNCING REQUEST FOR CANDIDATE ALGORITHM NOMINATIONS FOR THE ADVANCED ENCRYPTION STANDARD (AES). [Online] Available: http://csrc.nist.gov/encryption/aes/pre-round1/aes_9709.htm (Retrieved September 21, 1999)

National Institute of Standards and Technology (1998). AES Round 1 Information. [Online] Available: http://csrc.nist.gov/encryption/aes/round1/round1.htm#algorithms (Retrieved February 01, 2001)

National Institute of Standards and Technology (1999a). AES Round 2 Finalists. [Online] Available: http://csrc.nist.gov/encryption/aes/round2/round2.htm (Retrieved February 01, 2001)

National Institute of Standards and Technology (1999b). FIPS PUB 46-3 – Data Encryption Standard (DES), [Online] Available: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf (Retrieved May 21, 2000)

National Institute of Standards and Technology (2001). AES Algorithm (Rijndael) Information. [Online] Available: http://csrc.nist.gov/encryption/aes/rijndael/ (Retrieved October 11, 2001)

Nechvatal, J., Blaker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., & Roback, E. (2000). Report on the Development of the Advanced Encryption Standard (AES). [Online] Available: http://csrc.nist.gov/encryption/aes/round2/r2report.pdf (Retrieved October 18, 2000)

Needham, R., & Schroeder, M. (1978). Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM, 21(12)*, 993 - 999

Netscape Communications Corporation (1997a). Plug-in API Reference. [Online] Available: http://developer.netscape.com/docs/manuals/communicator/plugin/refov.htm#1017247 (Retrieved November 12, 2000)

Netscape Communications Corporation (1997b). Plug-in API Reference. [Online] Available: http://developer.netscape.com/docs/manuals/communicator/plugin/refov.htm (Retrieved November 12, 2000)

New Zealand Educational Software (2000). AudioGraph. [Online] Available: http://www.nzedsoft.com/audiographhomepa.html (Retrieved November 1, 2000).

Oliphant, Z. (2000). Plug-Ins for Netscape Navigator. [Online] Available: http://www.zdnet.com/pcmag/pctech/content/16/03/pp1603.001.html (Retrieved November 12, 2000)

Oppliger, R. (1999). Chapter 12: Copyright Protection in *Security Technologies for the World Wide Web*, 1999, Artech House Inc., 307 – 320

Pereira, R., & Adams, R. (1998). RFC 2451 – The ESP CBC-Mode Cipher Algorithms. [Online] Available: http://www.faqs.org/rfcs/rfc2451.html (Retrieved October 11, 1999)

Pfitzmann, B., & Schunter, M. (1996). Asymmetric Fingerprinting. *Proceedings of EUROCRYPT'96*, Springer-Verlag, 84 – 95.

Pfitzmann, B., & Waidner, M. (1997). Anonymous Fingerprinting. *Advances in Cryptology – EUROCRYPT'97*, Springer-Verlag, 88 – 102.

Popek, G.J., & Kline, C.S. (1979). Encryption and Secure Computer Networks. *ACM Computing Surveys, 11(4)*, 335--356

Pressman, R.S. (1997a). Chapter 16: Software Testing Techniques, *Software Engineering – A Practitioner's Approach* (pp. 448 – 486). McGraw-Hill.

Pressman, R.S. (1997b). Section 17.4: Integration Testing, *Software Engineering – A Practitioner's Approach* (pp. 498 – 503). McGraw-Hill.

Rijmen, V. (1999). The block cipher Rijndael. [Online] Available: http://www.esat.kuleuven.ac.be/~rijmen/rijndael/ (Retrieved February 5, 2001)

Rivest, R.L., Shamir, A., and Adleman, L.M. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM, v.21, n.2, Feb 1978*, 120 – 126.

Rivest, R.L., Shamir, A., and Adleman, L.M. (1979). On Digital Signatures and Public Key Cryptosystems. *MIT Laboratory for Computer Science*, Technical Report, MIT/LCS/TR-212, Jan 1979.

Roelofs, G. (1996a). A Basic Introduction to PNG Features. [Online] Available: http://www.libpng.org/pub/png/pngintro.html (Retrieved October 14, 2001)

Roelofs, G. (1996b). Portable Network Graphics. [Online] Available: http://www.libpng.org/pub/png/ (Retrieved November 25, 2000)

RSA Security Inc. (2001). RC6® Block Cipher. [Online] Available: http://www.rsasecurity.com/rsalabs/rc6/ (Retrieved February 5, 2001)

RSA Security Inc (2002). What are some other block ciphers? [Online] Available: http://www.rsasecurity.com/rsalabs/faq/3-6-7.html, (Retrieved March 2, 2002)

Schneier, B. (1994). Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). *Fast Software Encryption*, Cambridge Security Workshop Proceedings, Springer-Verlag, *1994*, 191-204.

Schneier, B. (1995). The GOST Encryption Algorithm. *Dr. Dobb's Journal, v.20, n.1, Jan 95*, 123-124.

Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. U.S.A: John Wiley & Sons Inc.

Schneier, B. (2001). The Blowfish Encryption Algorithm – one year later. [Online] Available: http://www.counterpane.com/bfdobsoyl.html (Retrieved December 12, 2001).

Shannon, C.E. (1949). Communication Theory of Secrecy Systems. *Bell System Technical Journal, V.28, n.4, 1949*, 656 – 715.

Shimizu, A. & Miyaguchi, S. (1987). Fast Data Encipherment Algorithm FEAL. *Transactions of IEICE of Japan, V.J70-D, n.7, Jul 87*, 1413 – 1423.

Smith, J.L. (1971). The Design of Lucifer, A Cryptographic Device for Data Communications. *IBM Research Report RC3326.*

Stallings, W. (1995a). Chapter 4: Public-Key Cryptogrphy, Section 4.2: The RSA Algorithm in *Network and Internetwork Security: Principles and Practice*. New Jersey, U.S.A.: Prentice-Hall Inc. pp.128

Stallings, W. (1995b). *Network and Internetwork Security: Principles and Practice.* New Jersey, U.S.A: Prentice-Hall Inc.

Stein, L. D. (1998). *Web Security – A Step-by-Step Reference Guide.* Massachusetts: Addison-Wesley.

Sun Microsystems Java Software Division (1998). X.509 Certificates and Certificate Revocation Lists (CRLs). [Online] Available: http://java.sun.com/products/jdk/1.2/docs/guide/security/cert3.html (Retrieved December 15, 2001).

The Electronic Frontier Foundation (1999). RSA Code-Breaking Contest Again Won by Distributed.Net and Electronic Frontier Foundation (EFF) – DES Challenge III Broken in Record 22 Hours. [Online] Available: http://www.eff.org/pub/Privacy/Crypto_misc/DESCracker/HTML/19990119_deschallenge3.html (Retrieved May 20, 2000)

Vaudenay, S. (1996). On the Weak Keys of Blowfish. *Fast Software Encryption: Third International Workshop, 1039*, Cambridge, UK, 21-23 February 1996. Springer-Verlag, 27-32.

Watanabe, H., & Kasami, T. (1997). A Secure Code for Recipient Watermarking against Conspiracy Attacks by All Users. *Information and Communications Security, 1997*, 414 – 423

Wheeler, D. J., & Needham, R. M. (1994). TEA, a Tiny Encryption Algorithm *Fast Software Encryption: Second International Workshop*, *1008*, December 1994. Springer-Verlag, 363-366.

Appendix A: Source Code of Methods to Perform Security Check

```
LRESULT CUserWnd::OnKeyMsg(WPARAM wParam, LPARAM lParam)
{
	RKey *protectionKey = (RKey *)wParam;
	long companyKey = protectionKey->m_companyKey;
	long lowerProtectionKey = protectionKey->m_lowerProtectionKey;
	long upperProtectionKey = protectionKey->m_upperProtectionKey;

	char *charPtr;
	BYTE *bytePtr;

	// internal TEA key for encryption production key
	char *keyForProduct = "www.NZEdSoft.com";

	// the "tea_productKey" will be used to produce Blowfish key for
	// encrypting the userID
	tea_block tea_productKey[2];
	tea_key teaKey;

	charPtr = keyForProduct;
	teaKey.k[0] = mgetBLong(charPtr);
	teaKey.k[1] = mgetBLong(charPtr);
	teaKey.k[2] = mgetBLong(charPtr);
	teaKey.k[3] = mgetBLong(charPtr);

	tea_productKey[0].v[0] = companyKey;
	tea_productKey[0].v[1] = lowerProtectionKey;
	tea_productKey[1].v[0] = upperProtectionKey;
	tea_productKey[1].v[1] = 0L;

	// key length is 2, and it will do encryption
	// after encryption, the tea_productKey is in encrypted format
	tea_ecb(tea_productKey, &teaKey, 2, 1);

	BLOWFISH_KEY bfKey;	// actual blowfish key

	BYTE input[11];		// store the user input string
```

```
// store the correct password string (in plaintext)
BYTE password[11];

int keyLen, inputLen, passwordLen;
LPTSTR pInput, pPassword;

// Blowfish key, total length is 56 bytes (448 bits)
BYTE key[56];
BYTE output[11];  /* the output of Blowfish
                     encryption over userID */

memset(input, '\0', 11);
memset(key, '\0', 56);
memset(output, '\0', 11);
memset(password, '\0', 11);

keyLen = 16;
bytePtr = key;

// assign key value to Blowfish key
mputBLong(bytePtr, tea_productKey[0].v[0]);
mputBLong(bytePtr, tea_productKey[0].v[1]);
mputBLong(bytePtr, tea_productKey[1].v[0]);
mputBLong(bytePtr, tea_productKey[1].v[1]);

//initialize blowfish key
if( blowfishKeyInit( &bfKey, (unsigned char *)key, keyLen )
                        != CRYPT_OK )
{
      AfxMessageBox("Initialize blowfish key failed" );
      exit(1);
}

BYTE storeUserId[8];          // for read and store userId string
BYTE storePassword[16];       /* for read and
                                 store password string */

//internal TEA key for store userId and password
char *keyForStore = "MasseyAudiograph";
```

```
tea_block tea_userId;                                    //userId
tea_block tea_password[2];                               //password
tea_key storeKey; //TEA key for store the userId and password

//convert the keyForStore into tea_key structure
charPtr = keyForStore;
storeKey.k[0] = mgetBLong(charPtr);
storeKey.k[1] = mgetBLong(charPtr);
storeKey.k[2] = mgetBLong(charPtr);
storeKey.k[3] = mgetBLong(charPtr);

char* pFileName =   "C:\\AepProduction.dat";

TRY{
      CFile keyFile( pFileName, CFile::modeCreate
                                | CFile::modeNoTruncate
                                | CFile::modeReadWrite
                                | CFile::typeBinary );

      memset(input, '\0', 11);
      memset(output, '\0', 11);
      memset(password, '\0', 11);
      if(keyFile.GetLength() != 0 ){
            keyFile.Read(storeUserId, 8);
            keyFile.Read(storePassword, 16);

            bytePtr = storeUserId;
            tea_userId.v[0] = mgetBLong(bytePtr);
            tea_userId.v[1] = mgetBLong(bytePtr);

            bytePtr = storePassword;
            tea_password[0].v[0] = mgetBLong(bytePtr);
            tea_password[0].v[1] = mgetBLong(bytePtr);
            tea_password[1].v[0] = mgetBLong(bytePtr);
            tea_password[1].v[1] = mgetBLong(bytePtr);

            //TEA decryption of the userId and password
            tea_ecb(&tea_userId, &storeKey, 1, 0);
            tea_ecb(tea_password, &storeKey, 2, 0);
```

```
        // assigne the decrypted userID back into storeUserId
        bytePtr = storeUserId;
        mputBLong(bytePtr, tea_userId.v[0]);
        mputBLong(bytePtr, tea_userId.v[1]);

        /* assign the decrypted
           password back into storePassword */
        bytePtr = storePassword;
        mputBLong(bytePtr, tea_password[0].v[0]);
        mputBLong(bytePtr, tea_password[0].v[1]);
        mputBLong(bytePtr, tea_password[1].v[0]);
        mputBLong(bytePtr, tea_password[1].v[1]);

        memcpy(input, storeUserId, 8);
        memcpy(password, storePassword, 11);

}

this->DoEncryption(&bfKey, input, output);
if(memcmp(output, password, 11)
   || keyFile.GetLength() == 0)
{
        memset(input, '\0', 11);
        memset(password, '\0', 11);
        memset(output, '\0', 11);

        CPasswdDlg aPasswdDlg(this->GetTopLevelParent());
        if(aPasswdDlg.DoModal()==IDOK)
        {
                //get the user ID
                inputLen = aPasswdDlg.m_userId.GetLength();
                if(inputLen > 8) {
                        AfxMessageBox("Too many plaintext \
                                        characters!");
                }
                pInput =
                        aPasswdDlg.m_userId.GetBuffer(inputLen);
                memcpy(input,pInput,inputLen);

                //encryption and radix-64 conversion
```

```
this->DoEncryption(&bfKey, input, output);

//get user password
passwordLen =
      aPasswdDlg.m_password.GetLength();
if(passwordLen > 11){
      AfxMessageBox("password too long");
      SetDisabled(true);
      _event.Release();
      return 1L;
}
pPassword =
  aPasswdDlg.m_password.GetBuffer(passwordLen);
memcpy(password, pPassword, passwordLen);

//check if the password is correct
if(memcmp(output, password, 11) == 0 ){
      SetDisabled(false);
      /* write encrypted userId
         and password into file */
      memset(storeUserId, '\0', 8);
      memset(storePassword, '\0', 16);

      memcpy(storeUserId, input, 8);
      memcpy(storePassword, password, 11);

      bytePtr = storeUserId;
      tea_userId.v[0] = mgetBLong(bytePtr);
      tea_userId.v[1] = mgetBLong(bytePtr);

      bytePtr = storePassword;
      tea_password[0].v[0] =
                  mgetBLong(bytePtr);
      tea_password[0].v[1] =
                  mgetBLong(bytePtr);
      tea_password[1].v[0] =
                  mgetBLong(bytePtr);
      tea_password[1].v[1] =
                  mgetBLong(bytePtr);
```

```
                        tea_ecb(&tea_userId, &storeKey, 1, 1);
                        tea_ecb(tea_password, &storeKey, 2, 1);

                        memset(storeUserId, '\0', 8);
                        memset(storePassword, '\0', 16);

                        bytePtr = storeUserId;
                        mputBLong(bytePtr, tea_userId.v[0]);
                        mputBLong(bytePtr, tea_userId.v[1]);

                        bytePtr = storePassword;
                        mputBLong(bytePtr, tea_password[0].v[0]);
                        mputBLong(bytePtr, tea_password[0].v[1]);
                        mputBLong(bytePtr, tea_password[1].v[0]);
                        mputBLong(bytePtr, tea_password[1].v[1]);

                        keyFile.SeekToBegin();
                        keyFile.Write(storeUserId, 8);
                        keyFile.Write(storePassword, 16);

                }
                else{
                        SetDisabled(true);
                        CProtectDlg aDlg;
                        aDlg.DoModal();
                }
        }
        else if(memcmp(output, password, 11) == 0){
                SetDisabled(false);
        }
        keyFile.Close();
}
CATCH( CFileException, e ){
        #ifdef _DEBUG
                afxDump << "File could not be opened "
                        << e->m_cause << "\n";
                AfxMessageBox("File could not be opened");
                //AfxMessageBox(pFilePath);
                exit(1);
```

```
            #endif
        }
        END_CATCH
        _event.Release();
        return 0L;
}

//////////////////////////////////////////////////////////////////
// DoEncryption(BLOWFISH_KEY *bfKey, BYTE *input, BYTE *output)
// bfKey: pointer to blowfish key
// input: pointer to data been encrypted, 8 bytes
// output: pointer to encrypted data after Radix-64 converting, 11 byte
//
void CUserWnd::DoEncryption(BLOWFISH_KEY *bfKey,
                            BYTE *input, BYTE *output)
{
        long cipherL = 0L, cipherR = 0L;
        BYTE tmpbuf[8];
        BYTE buffer[11];
        BYTE *bytePtr;

        memcpy(tmpbuf, input, 8);
        memset(buffer, '\0', 11);

        //blowfish encryption of input
        blowfishEncrypt( bfKey, tmpbuf );

        //prepare for Radix-64 encoding of tmpbuf
        bytePtr = tmpbuf;
        cipherL = mgetBLong(bytePtr);
        cipherR = mgetBLong(bytePtr);

        //convert the cipher into 6-bits group in the buffer
        buffer[ 10 ] = ( BYTE ) ( cipherR & 0x3F );
        cipherR = cipherR >> 6;
        buffer[ 9 ] = ( BYTE ) ( cipherR & 0x3F );
        cipherR = cipherR >> 6;
        buffer[ 8 ] = ( BYTE ) ( cipherR & 0x3F );
        cipherR = cipherR >> 6;
        buffer[ 7 ] = ( BYTE ) ( cipherR & 0x3F );
```

```
        cipherR = cipherR >> 6;
        buffer[ 6 ] = ( BYTE ) ( cipherR & 0x3F );
        cipherR = cipherR >> 6;

        buffer[ 5 ] = ( BYTE ) ( ( (cipherL & 0x0F) << 2 )
                                        | (cipherR & 0x03) );
        cipherL = cipherL >> 4;

        buffer[ 4 ] = ( BYTE ) ( cipherL & 0x3F );
        cipherL = cipherL >> 6;
        buffer[ 3 ] = ( BYTE ) ( cipherL & 0x3F );
        cipherL = cipherL >> 6;
        buffer[ 2 ] = ( BYTE ) ( cipherL & 0x3F );
        cipherL = cipherL >> 6;
        buffer[ 1 ] = ( BYTE ) ( cipherL & 0x3F );
        cipherL = cipherL >> 6;
        //only 4 bits are from cipher text
        buffer[ 0 ] = ( BYTE ) ( cipherL & 0x0F );

        //Radix-64 encoding
        for(int i=0; i<11; i++){
                output[i] = (BYTE) ( radixEncode(buffer[i]) );
        }
};
```

Appendix B: Source Code of Methods to Generate Password for a UserID

```
void CSetKeyDlg::OnChangeEditUserid()
{
        int i;

        BLOWFISH_KEY bfKey;

        //input user ID length & key length
        int userIdLen, keyLen;

        LPTSTR pUserId, pPassword;

        BYTE key[56];                   //blowfish key
        long cipherL = 0L, cipherR = 0L;//cipher data

        BYTE input[11];                 //for input plain text
        BYTE buffer[11];                //data to be encrypted
        BYTE output[11];                //for output text

        //retrieve data from dialog box
        UpdateData(true);

        //reset all the array
        memset(key, '\0', 56);
        memset(input, '\0', 11);
        memset(buffer, '\0', 11);
        memset(output, '\0', 11);

        //get the input text (user ID)
        userIdLen = m_userId.GetLength();
        if(userIdLen > 8) {
                AfxMessageBox("Too many plaintext characters!");
                return;
        }
        pUserId = m_userId.GetBuffer(userIdLen);

        //copy the userID from user into variable "input"
```

```
memcpy(input,pUserId,userIdLen);

//encryption the key using TEA encryption method
tea_block productKey[2];
tea_key teaKey;

//an internal TEA key string
char *keyStr = "www.NZEdSoft.com";

teaKey.k[0] = keyStr[0] << 24 | keyStr[1] << 16 |
              keyStr[2] << 8 | keyStr[3] << 0;

teaKey.k[1] = keyStr[4] << 24 | keyStr[5] << 16 |
              keyStr[6] << 8 | keyStr[7] << 0;

teaKey.k[2] = keyStr[8] << 24 | keyStr[9] << 16 |
              keyStr[10] << 8 | keyStr[11] << 0;

teaKey.k[3] = keyStr[12] << 24 | keyStr[13] << 16 |
              keyStr[14] << 8 | keyStr[15] << 0;

productKey[0].v[0] =  m_companyKey;
productKey[0].v[1] = m_lowerKey;
productKey[1].v[0] = m_upperKey;
productKey[1].v[1] = 0L;

// TEA encryption of productKey
tea_ecb(productKey, productKey, &teaKey, 2, 1);

//get the user key
keyLen = 16;

key[0] = (productKey[0].v[0] >> 24) & 0xFF;
key[1] = (productKey[0].v[0] >> 16) & 0xFF;
key[2] = (productKey[0].v[0] >> 8) & 0xFF;
key[3] = (productKey[0].v[0] >> 0) & 0xFF;

key[4] = (productKey[0].v[1] >> 24) & 0xFF;
key[5] = (productKey[0].v[1] >> 16) & 0xFF;
key[6] = (productKey[0].v[1] >> 8) & 0xFF;
```

```
key[7] = (productKey[0].v[1] >> 0) & 0xFF;

key[8] = (productKey[1].v[0] >> 24) & 0xFF;
key[9] = (productKey[1].v[0] >> 16) & 0xFF;
key[10] = (productKey[1].v[0] >> 8) & 0xFF;
key[11] = (productKey[1].v[0] >> 0) & 0xFF;

key[12] = (productKey[1].v[1] >> 24) & 0xFF;
key[13] = (productKey[1].v[1] >> 16) & 0xFF;
key[14] = (productKey[1].v[1] >> 8) & 0xFF;
key[15] = (productKey[1].v[1] >> 0) & 0xFF;

//test on TEA decryption
tea_ecb(productKey, productKey, &teaKey, 2, 0);
if(productKey[0].v[0] !=  m_companyKey ||
        productKey[0].v[1] != m_lowerKey ||
        productKey[1].v[0] != m_upperKey ||
        productKey[1].v[1] != 0L)
{
        AfxMessageBox("TEA Decryption error");
        exit(1);
}
//end of test on TEA decryption

//initialize blowfish key
if( blowfishKeyInit( &bfKey, (unsigned char *)key, keyLen )
                                                != CRYPT_OK )
{
        AfxMessageBox("Initialize blowfish key failed" );
        return;
}

//encryption of the input data (user ID)
blowfishEncrypt( &bfKey, input );

//prepare for Radix-64 encoding
cipherL =( ( ( LONG ) input[ 0 ] << 24 )        \
         | ( ( LONG ) input[ 1 ] << 16 ) \
         | ( ( LONG ) input[ 2 ] << 8 )  \
         | ( LONG ) input[ 3 ] );
```

```
cipherR =( ( ( LONG ) input[ 4 ] << 24 )        \
         | ( ( LONG ) input[ 5 ] << 16 ) \
         | ( ( LONG ) input[ 6 ] << 8 )  \
         | ( LONG ) input[ 7 ] );

//convert the cipher data into 6-bits group
buffer[ 10 ] = ( BYTE ) ( cipherR & 0x3F );
cipherR = cipherR >> 6;
buffer[ 9 ] = ( BYTE ) ( cipherR & 0x3F );
cipherR = cipherR >> 6;
buffer[ 8 ] = ( BYTE ) ( cipherR & 0x3F );
cipherR = cipherR >> 6;
buffer[ 7 ] = ( BYTE ) ( cipherR & 0x3F );
cipherR = cipherR >> 6;
buffer[ 6 ] = ( BYTE ) ( cipherR & 0x3F );
cipherR = cipherR >> 6;

buffer[ 5 ] = ( BYTE ) ( ( (cipherL & 0x0F) << 2 ) \
                       | (cipherR & 0x03) );
cipherL = cipherL >> 4;

buffer[ 4 ] = ( BYTE ) ( cipherL & 0x3F );
cipherL = cipherL >> 6;
buffer[ 3 ] = ( BYTE ) ( cipherL & 0x3F );
cipherL = cipherL >> 6;
buffer[ 2 ] = ( BYTE ) ( cipherL & 0x3F );
cipherL = cipherL >> 6;
buffer[ 1 ] = ( BYTE ) ( cipherL & 0x3F );
cipherL = cipherL >> 6;
//only 4 bits are from cipher text
buffer[ 0 ] = ( BYTE ) ( cipherL & 0x0F );

//Radix-64 encoding
for(i=0; i<11; i++){
        output[i] = (BYTE) ( radixEncode(buffer[i]) );
}

//display the output
pPassword = m_password.GetBuffer(11);
```

```
memcpy(pPassword, output, 11);

//blowfish decryption test, only for test
//definition of variable for decoding
BYTE dInput[11], dOutput[11], dBuffer[11], dPlain[11];
BYTE *dPtr;
long dCipherL=0L, dCipherR=0L;
long dPlainL=0L, dPlainR=0L;

memset(dInput, '\0', 11);
memset(dOutput, '\0', 11);
memset(dBuffer, '\0', 11);
memset(dPlain, '\0', 11);

//get the input cipher data
memcpy(dInput, pPassword, 11);

//convert the data in Radix-64 format to 6-bits format
for(i=0; i<11; i++){
        dBuffer[i] = (BYTE) (radixDecode(dInput[i]));
}

//join the 6-bits format cipher data together
dCipherL = (dBuffer[0] & 0x0F);
for(i=1; i<5; i++){
        dCipherL = (dCipherL << 6);
        dCipherL = (dCipherL | (dBuffer[i] & 0x3F));
}
dCipherL = (dCipherL << 4);
dCipherL = (dCipherL | ( (dBuffer[5] >> 2) & 0x0F ));

dCipherR = (dBuffer[5] & 0x03);
for(i=6; i<11; i++){
        dCipherR = (dCipherR << 6);
        dCipherR = (dCipherR | (dBuffer[i] & 0x3F) );
}

//put the cipher data into buffer for decoding
dPtr = dOutput;
mputBLong(dPtr, dCipherL);
```

```
        mputBLong(dPtr, dCipherR);

        //decryption
        blowfishDecrypt( &bfKey, dOutput );

        //test if decryption is correct
        memcpy(dPlain, pUserId, userIdLen);
        if(memcmp(dOutput, dPlain, 8)){
                AfxMessageBox("Decryption incorrect!");
                exit(1);
        }
        //end of blowfish decryption test

        //update dialog box (variable --> control interface)
        UpdateData(false);

        //release the buffer
        m_userId.ReleaseBuffer();
        m_password.ReleaseBuffer();
};
```