10 Data Hiding and Obfuscation

On-line lecture: http://buchananweb.co.uk/adv/unit04.html

10.1 Objectives

The key objectives of this unit are to:

- Outline obfuscation methods.
- Define methods used to encode data in order to hide its original content.
- Understand encryption methods used to hide data, and possible methods to overcome this obfuscation.
- Define how file types can be discovered.

10.2 Introduction

This unit provides an outline in some of the methods that a suspect may use to hide their tracks. Hiding information has existed for many decades in many different forms. In fact stenanography, which is the science of hiding information within content, has been arrange for thousands of years, and includes using invisible inks and to hide information. Another method of hiding information is to embed it into messages, such as in:

```
Let everyone tango. This has Edward's mind in
some simple inquiry of nothing, before everyone
gets into Nirvana.
```

which, when each of the starting characters is taken, gives the message of **Let the mission begin**. This type of hidden information is known as a covert channel where information is added through a communications channel which is was not intended for. Other covert channels have included, in the past: Passing a briefcase in a busy place; Hiding microfilms in objects; and using templates for typewritten text. Unfortunately as we move into the Information Age, the places that covert channels can exist increases by the day and it can often be difficult to detect this type of communication in electronic transmissions.

Figure 10.1 shows the main classifications for information hiding, including the use of:

- **Covert channels**. This is used a communication channel for a purpose that it was not intended for (Llamas, 2004).
- **Steganography**. This is the methods used to hide information in content that only the recipient knows where to look for the hidden information.
- **Anonymity**. This is the methods used to hide the original source of the information.
- **Copyright marking**. This typically involves embedded information, normally which is hidden with content.

The requirements for copyright marking is obviously a growing issue, as many content creators, such as musicians, artists, and so on, are keen to preserve their copyright on content. It is, though, a constant challenge, as many methods of copyrighting are normally flawed in some sort of way that means that copyright protection can often be overcome. The challenge is sometimes to preserve the copyright in some way which is invisible to the user, but can be revealed when required.

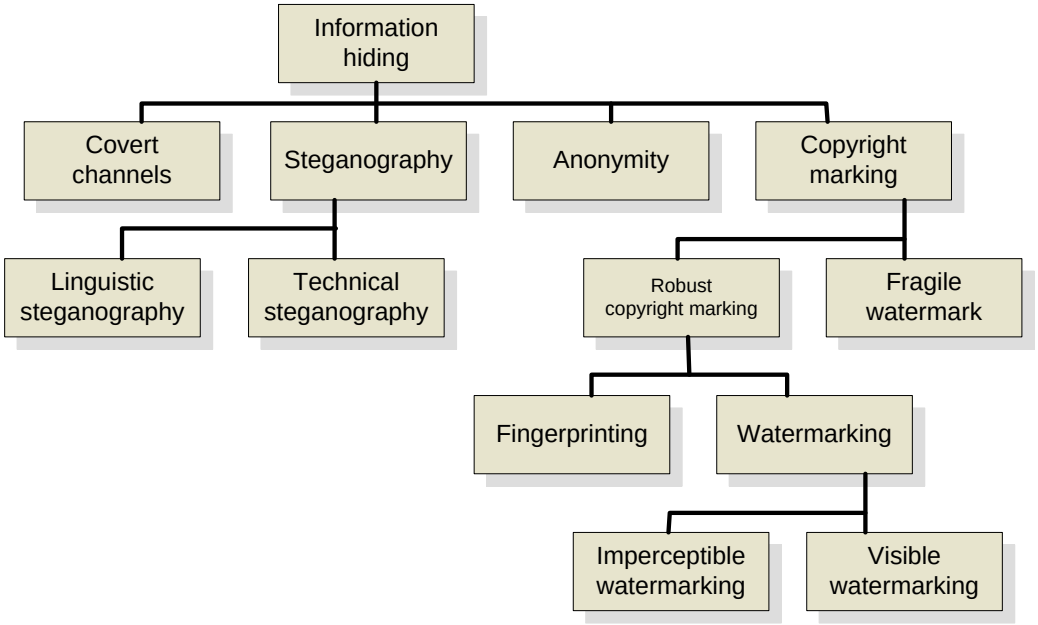


Figure 10.1 Information hiding classifications

10.3 Obfuscation using Encryption

One method of obfuscating data is to encrypt or to encode the data in a non-clear format. The main methods that can be used include:

- **Private key encryption**. With private key a secret key is used to encrypt the data. To decrypt the original key must be found. Normally, though, the encryption key is generated through a password generate program, thus the range of actual encryption keys used can thus be limited to a search of well-known phrases. Typical private-key encryption methods are DES, 3DES and AES.
- **Public key encryption**. With public-key encryption the data is encrypted with one key (normally the public key) and a private key is used to decrypt the ciper-text. A typical public-key method is RSA.
- **Hashing**. Hashing normally involves a one-way hashing function, where it is difficult to reverse the hashing. Some form of dictionary lookup is normally used to try and determine the original data.
- **Encoding**. This normally involves obfuscating messages by converting them into a non-readable format. Typical methods used include converting in Base-64 and also using an X-OR with a passphrase.

Private-key data hiding

Private key involves using the same key to encrypt as to decrypt. It is often used in encryption is it is fairly fast, and it does not need the same processing power of public key encryption. It can thus be supported on a wide range of devices. The typical ways to decrypt private-key encryption is:

- **Search for key strings.** With this method a scan is made of the host machine to find all the string that have been used in other types of access, such as for Internet Explorer passwords. These are the most likely ones that could lead to a successful decrypt.
- **Use a dictionary**. The next quickest method to find an encryption key is to use a standard dictionary to determine the key that it data has been encrypted with.
- **Perform a brute-force.** If the first two methods fail, a brute force can be conducted which will search through the key space.

In Figure 10.2 it can be seen that a word named "fred" has been encrypted with the key word of "apples", to produce a cipertext of "2A699…A04". A search is then conducted from words in the dictionary, where an exception is caused if the encryption process fails. This results in a number of possible encryption keys. In this case, these are "anyway", "apples", "assembler", and so on. It can be seen that "apples" is the only one which produces a sensible decryption.

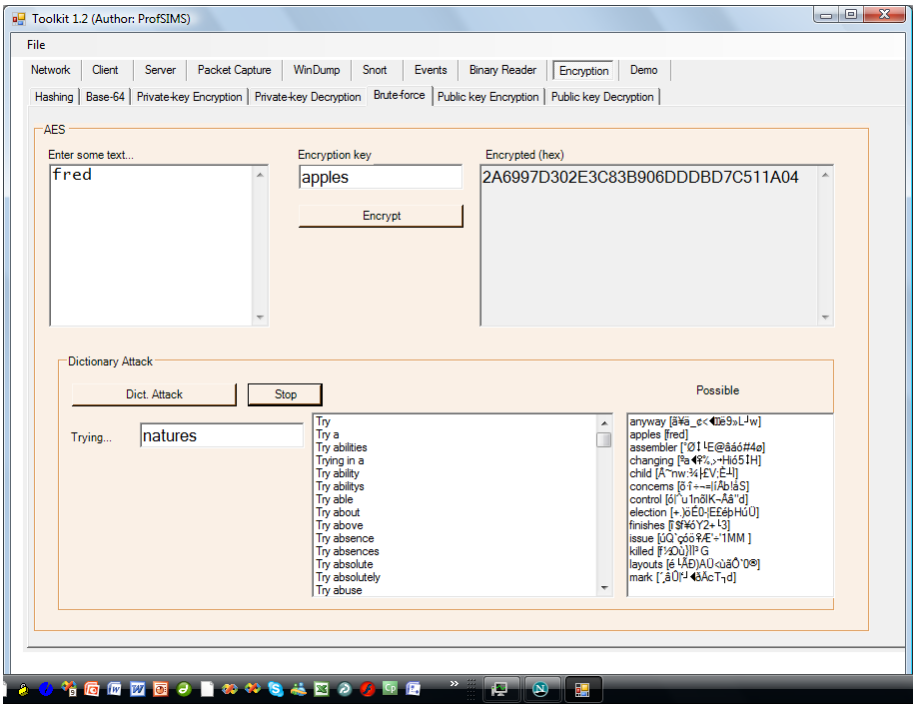


Figure 10.2 Dictionary search

http://buchananweb.co.uk/adv_security_and_network_forensics/dotnetclient_brute/dotnetclient_brute.htm

Public-key data hiding

Public key methods, such as with RSA, involve a different decryption key from the encryption one. These are known as a key pair. The key sizes tend to be fairly large as compared with private key methods (typically more than 1,024 bits, as apposed to 128/256 bit sizes for AES). It is thus extremely difficult to perform a brute force attack on the private key. The normal method is to try and determine the digital certificate which stores the public key and the private key. In Figure 10.3, it can be seen that the certificate on the left-hand side only contains the public key, whereas the one of the right-hand side contains both the public and the private key. Normally this certificate is protected by a password, thus the certificate can be opened using a dictionary or brute force search. A typical format for a certificate with a password is PFX.

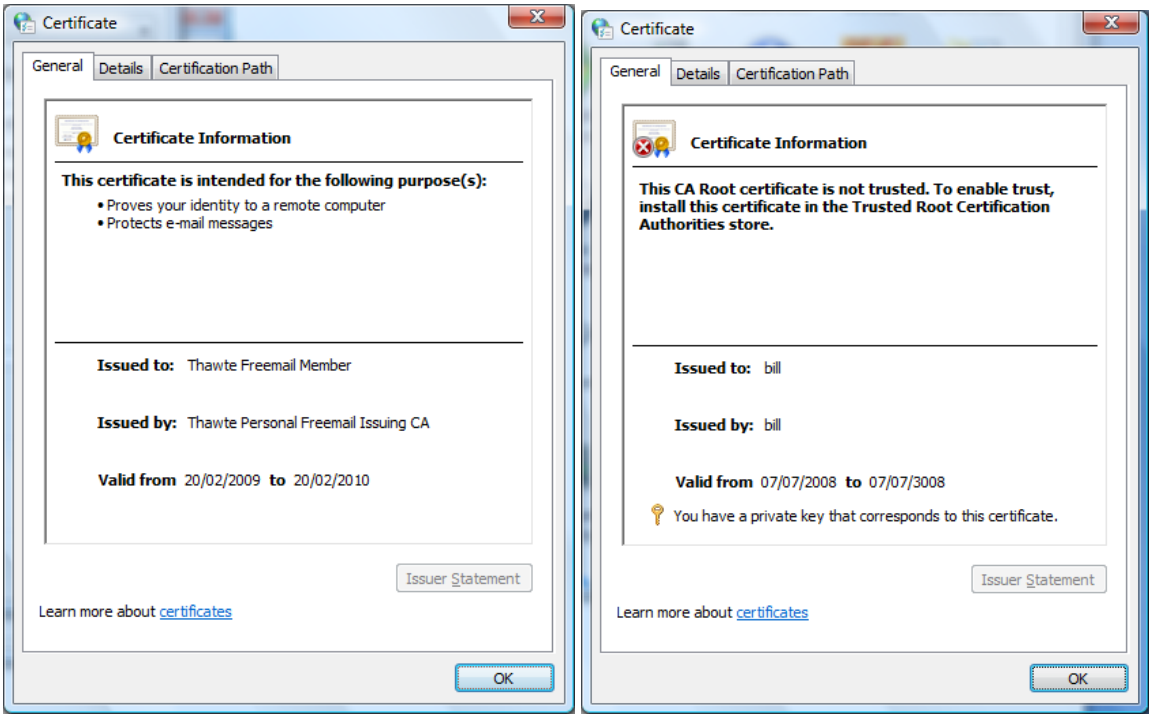


Figure 10.3 Digital certificates

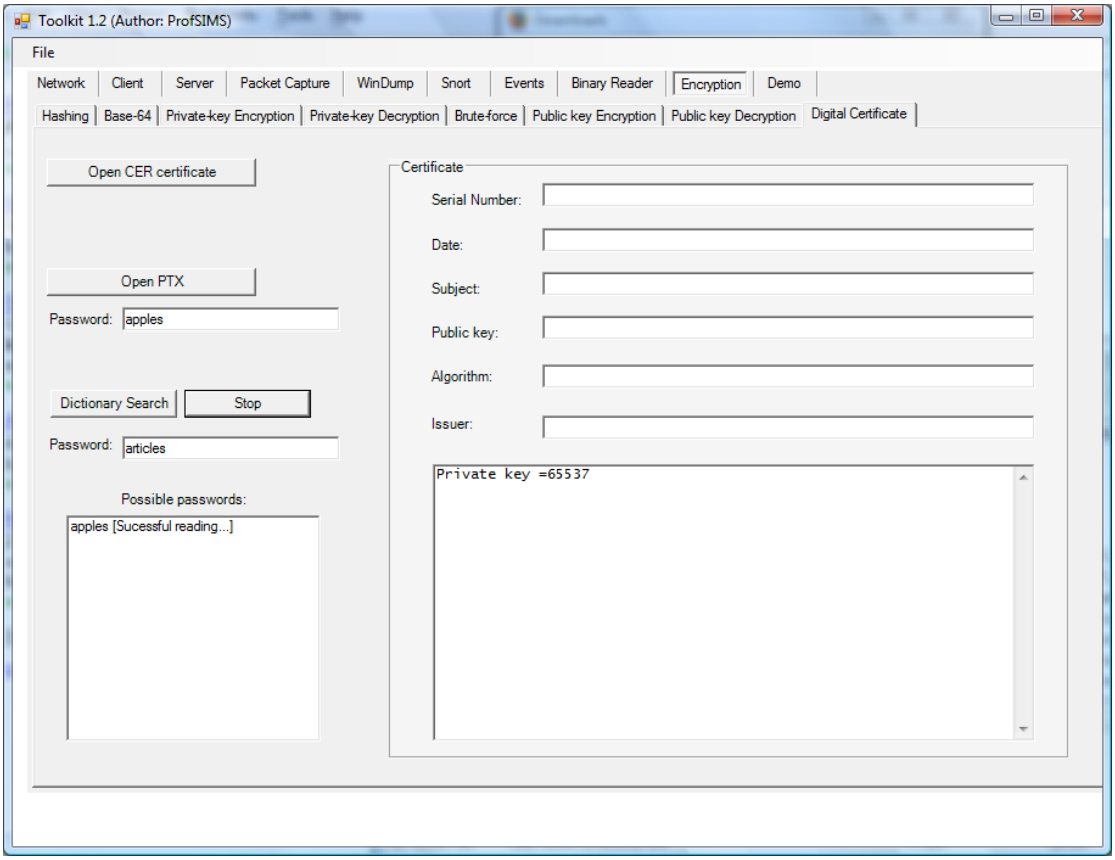


Figure 10.4 Searching for a password on a certificate

http://buchananweb.co.uk/adv_security_and_network_forensics/dotnetclient_digitalcert/dotnetclient_digitalcert.htm

Hashing

Hashing can be used to store messages, using a one-way encryption process. It is almost impossible to determine the original message from a hashed version, unless there is a dictionary for well-known hash functions. For example, "test" gives:

098F6BCD4621D373CADE4E832627B4F6

In this way a secret message can be kept in a hash format. A way to change the hash is to apply salt, where the hash varies based on a number of known keywords. For example:

Password="test";
Salt=One of {"fred","bert","ken"}
Hash = md5(Password.Salt);

The mdcrack program can be used to reverse the process, such as:

```
C:\test> mdcrack --algorithm=MD5 098F6BCD4621D373CADE4E832627B4F6

System / Starting MDCrack v1.8(3)
System / Running as mdcrack-sse --algorithm=MD5 098F6BCD4621D373CADE4E832627B4F6
```

```
System / Charset is: abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVW
XYZ
System / Detected processor(s): 2 x INTEL Itanium | MMX | SSE | SSE2 | SSE3
System / Target hash: 098F6BCD4621D373CADE4E832627B4F6
System / >> Using MD5 cores: maximal candidate/user salt size: 16/54 bytes
Info   / Press ESC for available runtime shortcuts (Ctrl-c to quit)
Info   / Thread #0: >> Using Core 1
Info   / Thread #0: Candidate size:  1 ( + user salt: 0 )
Info   / Thread #0: Candidate size:  2 ( + user salt: 0 )
Info   / Thread #0: Candidate size:  3 ( + user salt: 0 )
Info   / Thread #1: >> Using Core 1
Info   / Thread #1: Candidate size:  1 ( + user salt: 0 )
Info   / Thread #1: Candidate size:  2 ( + user salt: 0 )
Info   / Thread #1: Candidate size:  3 ( + user salt: 0 )
Info   / Thread #0: Candidate size:  4 ( + user salt: 0 )
Info   / Thread #1: Candidate size:  4 ( + user salt: 0 )
-----------------------------------------------------------/ Thread #1 (Success)
\----
System / Thread #1: Collision found: test
Info   / Thread #1: Candidate/Hash pairs tested: 2 341 902 ( 2.34e+006 ) in 812m
s
Info   / Thread #1: Allocated key space: 2.42e+028 candidates, 0.00% done
Info   / Thread #1: Average speed: ~ 2 884 116 ( 2.88e+006 ) h/s
```

This takes less than two seconds to run, while longer text sequences take much longer.

Encoding

There are many standard for encoding data from one format to another. One of the most common is Base-64, which is used to convert from an 8-bit format into 6-bit values, which are converted to Base-64 characters. The table for the conversion is given in Table 4.1.

Figure 10.1 Base-64 conversion

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

For example:

“What’

Results in:

“ 00100010
W 01010111

h	01101000
a	01100001
t	01110100
‘	00100111

Which gives: `0010001001010111011010000110000101110100001001110 1`
`001000 100101 011101 101000 011000 010111 010000 100111`
`I l d o Y X Q n`

The conversion thus becomes:

ASCII: "What's in a name? That which we call a rose. By any other name would smell as sweet."

Base-64: IldoYXQncyBpbiBhIG5hbWU/IFRoYXQgd2hpY2ggd2UgY2Fs bCBhIHJvc2UuIEJ5IGFueSBvdGhlciBuYW1lIHdvdWxkIHNtZ WxsIGFzIHN3ZWV0LiI=

Hex: 225768617427732069 6E2061206E616D653F20546861742077686963682077652063616C6C206120726F73652E20427920616E79206F74686572206E616D6520776F756C6420736D656C6C206173207377656574E22

Binary: 00100010 … 000010111000100010

http://buchananweb.co.uk/adv_security_and_network_forensics/dotnet_base64/dotnet_base64.htm

Ex-OR encoding

The Ex-OR operator is used in many applications in data hiding and encryption, especially as it does not lose any information within the bit stream. Its basic operation is:

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

The main advantage of Ex-OR is that a bit stream when Ex-OR’ed with a given value will result in the same value when it is Ex-OR’ed again. For example, if the text message is “Hello”, then the bit stream will be:

H	e	l	l	o
01001000	01100101	01101100	01101100	01101111

If we Ex-OR this will a bitvalue of 0101 0101 (‘U’) we get:

01001000	01100101	01101100	01101100	01101111
01010101	01010101	01010101	01010101	01010101
00011101	00110000	00111001	00111001	00111010

(1D, 30, 39, 3A)

And if we Ex-OR this with the same value:

00011101	00110000	00111001	00111001	00111010
01010101	01010101	01010101	01010101	01010101
01001000	01100101	01101100	01101100	01101111

Which results in the original value.

http://buchananweb.co.uk/adv_security_and_network_forensics/dotnet_xor/dotnet_xor.htm

Coding

There are obviously an infinite amount of ways that someone can hide or pass secret information using their own standard codes. Alphabet shifting is an example of this, where the alphabet is shifted by a given number of space, such as for a three letter shift:

Input: `abcdefghijklmnopqrstuvwxyz`
Output: `DEFGHIJKLMNOPQRSTUVWXYZABC`

Where "fred" would give "IUHG". Unfortunately this type of code is relatively easy to crack, as there are only 25 unique mappings. A more robust code is to randomly assign the letters, such as for:

```
In Chapter 1 the concept of defence-in-depth was discussed, where a defence system has many layers of defence. Unfortunately, as in military systems, it is not always possible to protect using front-line [… text missed out …] where intrusion detection agents are used to listen to network traffic, and network/user activity to try and detect any breaches in security.
```

And using the mapping of:

Code A

```
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
B I H O Q K W C D V L E J S R G X F A U T M Y N P Z
```

Becomes (Figure 10.5):

DS HCBGUQF 1 UCQ HRSHQGU RK OQKQSHQ-DS-OQGUC YBA ODAHTAAQO, YCQFQ B OQKQSHQ APAUQJ CBA JBSP EBPQFA RK OQKQSHQ. TSKRFUTS-BUQEP, BA DS JDEDUBFP APAUQJA, DU DA SRU BEYBPA GRAADIEQ UR GFRUQHU TADSW KFRSU-EDSQ OQKQSHQA, QMQS DK UCQFQ BFQ JTEUDGEQ EBPQFA RK UCQJ, BWBDSAU IFQBHCQA DS AQHTFDUP (KDWTFQ 2.2). UCDA HBS IQ IQHBTAQ BS DSUFTOQF CBA KRTSO B YQBLSQAA YDUCDS UCQ [… text missed out …] HRSHQGU, YCQFQ DSUFTADRS OQUQHUDRS BWQSUA BFQ TAQO UR EDAUQS UR SQUYRFL UFBKKDH, BSO SQUYRFL/TAQF BHUDMDUP UR UFP BSO OQUQHU BSP IFQBHCQA DS AQHTFDUP.

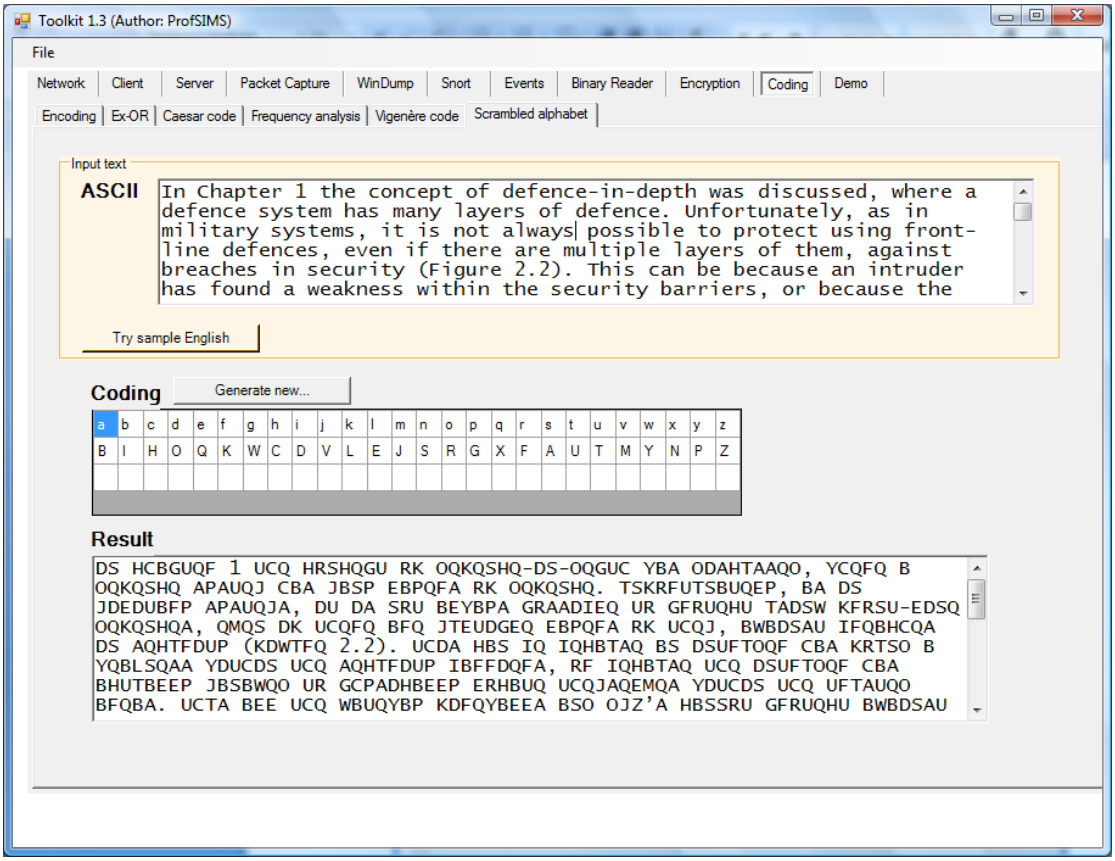


Figure 10.5 Searching for a password on a certificate

In standard English text, some letters are more probable than others, such the most popular is "E", and the least popular is "Z". In the following the coded text probability from the previous example has been mapped to the most probable letters to give (Figure 10.6):

Code B

E T O A N I R S H D L C F U M P Y W G B V K X J Q Z
q u b a s d f r c h e p t o k y w j g l m i z x v n

If we refer to the before, then the 'q' is the most popular letter, which has successfully determined the mapping (see Code A). The next most popular letter is a 'u', which maps to a 'T', which again is correct and just with these two letters gives:

DS HCBGteF 1 tCe HRSHeGt RK OeKeSHe-DS-OeGtC YBA ODAHTAAeO

After this it is normally a matter of moving the letters around, and identifying common words. For example, it can be see that the four work is likely to be “The”, thus a “C” could map to an “h” to give:

DS HhBGteF 1 the HRSHeGt RK OeKeSHe-DS-OeGtC YBA ODAHTAAeO

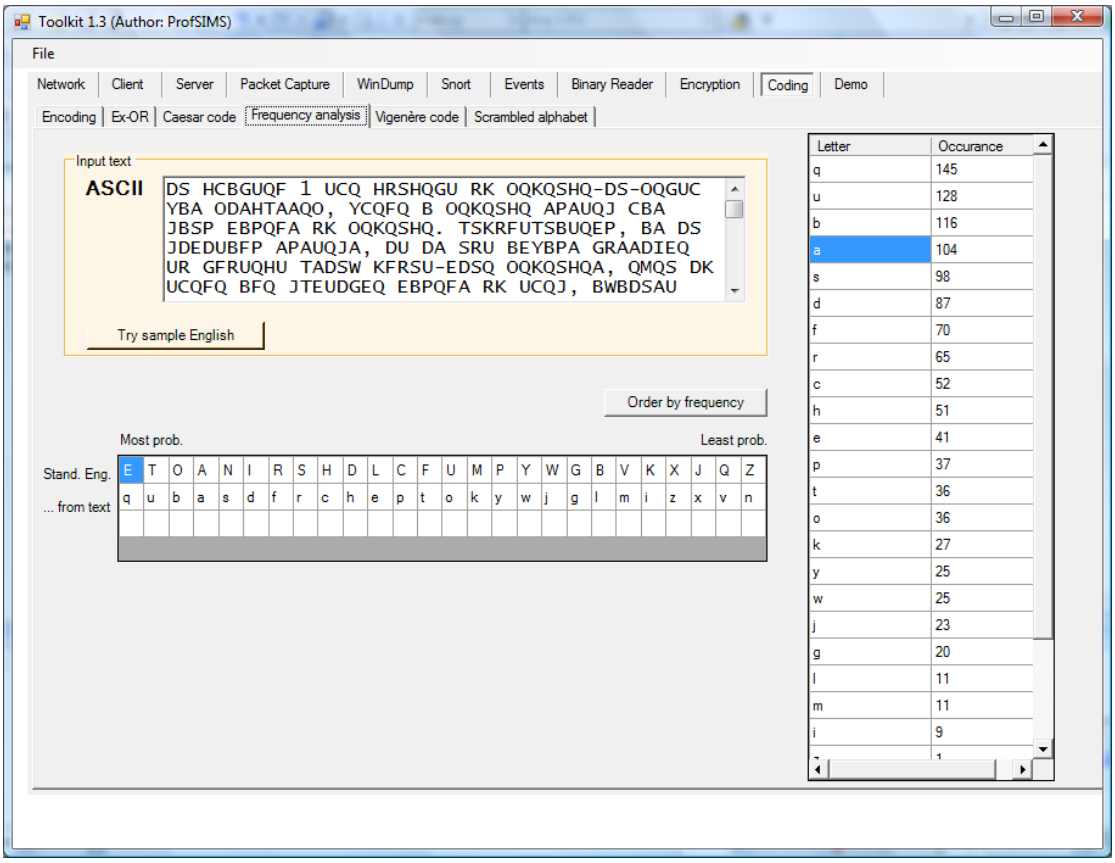


Figure 10.6 Statistical analysis

10.4 Obfuscation through tunneling

One method used to hide communications is to tunnel the information either through an encryption tunnel, or through another protocol. For an encryption tunnel the two ends of the tunnel negitate their encryption keys, and the communications will then be encrypted for the session. Thus any listening devices will not be able to decrypt the content, as they do not have the encryption keys required to decrypt the message. The two main methods used to create a tunnel are IPSec and SSL. With IPSec the start of the connection is identified with a connection on UDP Port 500, such as:

```
No.      Time          Source                Destination           Protocol Info
      6 5.007300     192.168.0.20          146.176.210.2         ISAKMP   Aggressive

User Datagram Protocol, Src Port: 65341 (65341), Dst Port: isakmp (500)
    Source port: 65341 (65341)
    Destination port: isakmp (500)
```

```
    Length: 884
    Checksum: 0xa205 [correct]
        [Good Checksum: True]
        [Bad Checksum: False]
Internet Security Association and Key Management Protocol
    Initiator cookie: 0490174339C81264
    Responder cookie: 0000000000000000
    Next payload: Security Association (1)
    Version: 1.0
    Exchange type: Aggressive (4)
    Flags: 0x00
        .... ...0 = Not encrypted
        .... ..0. = No commit
        .... .0.. = No authentication
    Message ID: 0x00000000
    Length: 860
    Security Association payload
        Next payload: Key Exchange (4)
        Payload length: 556
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload # 1
            Next payload: NONE (0)
            Payload length: 544
            Proposal number: 1
            Protocol ID: ISAKMP (1)
            SPI Size: 0
            Proposal transforms: 14
            Transform payload # 1
                Next payload: Transform (3)
                Payload length: 40
                Transform number: 1
                Transform ID: KEY_IKE (1)
                Encryption-Algorithm (1): AES-CBC (7)
                Hash-Algorithm (2): SHA (2)
                Group-Description (4): Alternate 1024-bit MODP group (2)
                Authentication-Method (3): XAUTHInitPreShared (65001)
                Life-Type (11): Seconds (1)
                Life-Duration (12): Duration-Value (2147483)
                Key-Length (14): Key-Length (256)
    Key Exchange payload
        Next payload: Nonce (10)
        Payload length: 132
        Key Exchange Data (128 bytes / 1024 bits)
    Nonce payload
        Next payload: Identification (5)
        Payload length: 24
        Nonce Data
    Identification payload
        Next payload: Vendor ID (13)
        Payload length: 24
        ID type: 11
        ID type: KEY ID (11)
        Protocol ID: UDP (17)
        Port: 500
        Identification Data
    Vendor ID: draft-beaulieu-ike-xauth-02.txt
        Next payload: Vendor ID (13)
        Payload length: 12
        Vendor ID: draft-beaulieu-ike-xauth-02.txt
    Vendor ID: RFC 3706 Detecting Dead IKE Peers (DPD)
        Next payload: Vendor ID (13)
        Payload length: 20
        Vendor ID: RFC 3706 Detecting Dead IKE Peers (DPD)
    Vendor ID: Cisco Fragmentation
        Next payload: Vendor ID (13)
        Payload length: 24
        Vendor ID: Cisco Fragmentation
    Vendor ID: draft-ietf-ipsec-nat-t-ike-02\n
        Next payload: Vendor ID (13)
        Payload length: 20
        Vendor ID: draft-ietf-ipsec-nat-t-ike-02\n
    Vendor ID: CISCO-UNITY-1.0
        Next payload: NONE (0)
        Payload length: 20
        Vendor ID: CISCO-UNITY-1.0
```

```
No.     Time        Source                Destination           Protocol Info
      7 5.312130    146.176.210.2         192.168.0.20          ISAKMP   Aggressive
User Datagram Protocol, Src Port: isakmp (500), Dst Port: 65341 (65341)
    Source port: isakmp (500)
    Destination port: 65341 (65341)
    Length: 456
    Checksum: 0x5907 [correct]
        [Good Checksum: True]
        [Bad Checksum: False]
Internet Security Association and Key Management Protocol
    Initiator cookie: 0490174339C81264
    Responder cookie: F4B6486D172C028B
    Next payload: Security Association (1)
    Version: 1.0
    Exchange type: Aggressive (4)
    Flags: 0x00
        .... ...0 = Not encrypted
        .... ..0. = No commit
        .... .0.. = No authentication
    Message ID: 0x00000000
    Length: 448
    Security Association payload
        Next payload: Key Exchange (4)
        Payload length: 56
        Domain of interpretation: IPSEC (1)
        Situation: IDENTITY (1)
        Proposal payload # 1
            Next payload: NONE (0)
            Payload length: 44
            Proposal number: 1
            Protocol ID: ISAKMP (1)
            SPI Size: 0
            Proposal transforms: 1
            Transform payload # 10
                Next payload: NONE (0)
                Payload length: 36
                Transform number: 10
                Transform ID: KEY_IKE (1)
                Encryption-Algorithm (1): 3DES-CBC (5)
                Hash-Algorithm (2): MD5 (1)
                Group-Description (4): Alternate 1024-bit MODP group (2)
                Authentication-Method (3): XAUTHInitPreShared (65001)
                Life-Type (11): Seconds (1)
                Life-Duration (12): Duration-Value (2147483)
    Key Exchange payload
        Next payload: Nonce (10)
        Payload length: 132
        Key Exchange Data (128 bytes / 1024 bits)
    Nonce payload
        Next payload: Identification (5)
        Payload length: 24
        Nonce Data
    Identification payload
        Next payload: Hash (8)
        Payload length: 12
        ID type: 1
        ID type: IPV4_ADDR (1)
        Protocol ID: UDP (17)
        Port: Unused
        Identification data: 146.176.210.2
    Hash payload
        Next payload: Vendor ID (13)
        Payload length: 20
        Hash Data
    Vendor ID: CISCO-UNITY-1.0
        Next payload: Vendor ID (13)
        Payload length: 20
        Vendor ID: CISCO-UNITY-1.0
    Vendor ID: draft-beaulieu-ike-xauth-02.txt
        Next payload: Vendor ID (13)
        Payload length: 12
        Vendor ID: draft-beaulieu-ike-xauth-02.txt
    Vendor ID: RFC 3706 Detecting Dead IKE Peers (DPD)
        Next payload: Vendor ID (13)
        Payload length: 20
```

```
        Vendor ID: RFC 3706 Detecting Dead IKE Peers (DPD)
    Vendor ID: draft-ietf-ipsec-nat-t-ike-02\n
        Next payload: NAT-D (draft-ietf-ipsec-nat-t-ike-01 to 03) (130)
        Payload length: 20
        Vendor ID: draft-ietf-ipsec-nat-t-ike-02\n
    NAT-D (draft-ietf-ipsec-nat-t-ike-01 to 03) payload
        Next payload: NAT-D (draft-ietf-ipsec-nat-t-ike-01 to 03) (130)
        Payload length: 20
        Hash of address and port: A9D9C6CAEA2D34812E57F925DC636F98
    NAT-D (draft-ietf-ipsec-nat-t-ike-01 to 03) payload
        Next payload: Vendor ID (13)
        Payload length: 20
        Hash of address and port: 4F38ED224B394682D4F05FF14D6F34AF
    Vendor ID: Microsoft L2TP/IPSec VPN Client
        Next payload: Vendor ID (13)
        Payload length: 24
        Vendor ID: Microsoft L2TP/IPSec VPN Client
    Vendor ID: 0171EF70172D028B237401446015B2D0
        Next payload: Vendor ID (13)
        Payload length: 20
        Vendor ID: 0171EF70172D028B237401446015B2D0
    Vendor ID: 1F07F70EAA6514D3B0FA96542A500407
        Next payload: NONE (0)
        Payload length: 20
        Vendor ID: 1F07F70EAA6514D3B0FA96542A500407Covert channels
```

It is through this phase that the main encryption parameters are negotiated.

10.5 Covert channels

A covert channel is a communication channel that allows two cooperating processes to transfer information in a manner that violates the system's security policy (Berg 1998). It is thus a way of communicating which is not part of the original design of the system, but can be used to transfer information to a process or user that, a priori, would not be authorised to access to that information. Covert channels only exist in systems with multilevel security (Proctor and Neumann 1992), which contain and manage information with different sensitivity levels. This it allows different users to access to the same information, at the same time, but from different points-of-view, depending on their requirements to know and their access privileges. The covert channel concept was introduced in 1973 (Lampson 1973), and are now generally, classified based on (Gligor 1993):

- **Scenarios**. In general, when building covert channels scenarios, there is a differentiation between storage and timing covert channels (Lipner 1975). Storage covert channels are where one process uses direct (or indirect) data writing, whilst another process reads the data. It generally uses a finite system resource that is shared between entities with different privileges. Covert timing channels use the modulation of certain resources, such as the CPU timing, in order to exchange information between processes.
- **Noise**. As with any other communication channel, covert channels can be noisy, and vary in their immunity to noise. Ideally, a channel immune to noise is one where the probability of the receiver receiving exactly what the sender has transmitted is unity, and there are no interferences in the transmission. Obviously, in real-life, it is very difficult to obtain these perfect channels, hence it is common to apply error correction codes, which can obviously reduce the bandwidth of the channel.

- **Information flows**. With conventional lines of transmission, different techniques are applied to increase the bandwidth. A similar method can be achieved in the covert channels. Channels where several information flows are transmitted between sender and receiver are denominated aggregated channels, and depending on how sent variables are initialized, read and reset, aggregations can be classified as serial, parallel, and so on. Channels with a unique information flow are denominated non-aggregated.

The concern for the presence of covert channels is common in high security systems (Figure 8.2), such as military ones, where typically two observed users know that someone wishes to listen to their conversations. Many of the studies done about attacks based on covert channels and its prevention have been done by US government and military bodies, such as the National Security Agency, US Air Force, National Computer Security Centre, and so on. However, in other environments it is also possible the existence of covert channels, especially in protocols like the TCP/IP protocol suite (Route 1996; Rowland 1996).

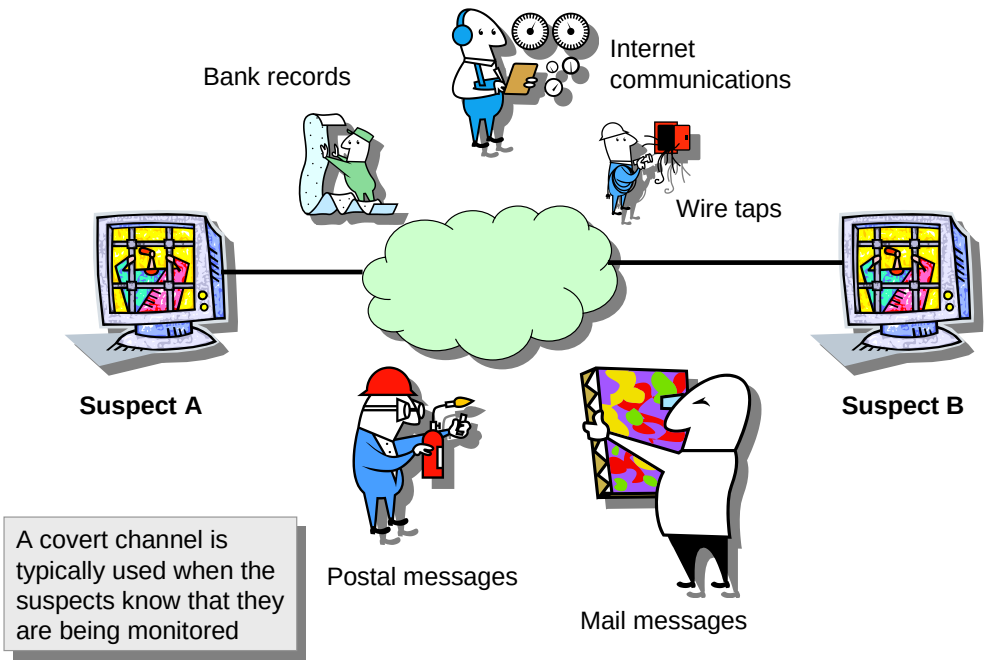


Figure 10.7 Covert channels

In covert channel scenarios *Alice* is often considered to be an inmate of a high security prison. It is assumed that she knows an escape plan from a prison where *Bob* is spending his sentence. *Alice* is trying to send the escape plan to *Bob*, however *Eve*, the governor checks their communication very precisely, thus they employ covert channel know to them to sent the secret messages (Kwecka, 2006). Figure 8.2 illustrates this.

IP and TCP data hiding

The IP and TCP protocols have many fields which are not actually necessary for most types of transmission. They could thus be a source of covert channels, as the additional fields are typically not checked by any intermediate device. In Figure 8.3 the

fields which could contain a covert channel in the IP header includes: Identification; TTL and Fragment Offset. For the Identification, the original RFC (RFC 791) defines that it ensures that the IP data packets have a unique identification number within a given time window. The implementation of the actual generation of the identification numbers has thus been left to the operation system developments. An example from Ubuntu shows that it starts with a random and then takes a jump after a given number of TCP segments:

```
No.     Time        Source                Destination           Protocol Info
     42 23.937372    192.168.75.138          192.168.75.1            TCP       54064 >
icslap [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=18836 TSER=0 WS=5
    Identification: 0x1640 (5696)

No.     Time        Source                Destination           Protocol Info
     44 23.943145    192.168.75.138          192.168.75.1            TCP       54064 >
icslap [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSV=18838 TSER=2182531
    Identification: 0x1641 (5697)

No.     Time        Source                Destination           Protocol Info
     45 23.945922    192.168.75.138          192.168.75.1            TCP       54064 >
icslap [PSH, ACK] Seq=1 Ack=1 Win=5856 Len=133 TSV=18838 TSER=2182531
    Identification: 0x1642 (5698)

No.     Time        Source                Destination           Protocol Info
     49 23.974294    192.168.75.138          192.168.75.1            TCP       54064 >
icslap [ACK] Seq=134 Ack=225 Win=6912 Len=0 TSV=18845 TSER=2182534

    Identification: 0x1643 (5699)

No.     Time        Source                Destination           Protocol Info
     50 23.974900    192.168.75.138          192.168.75.1            TCP       54064 >
icslap [ACK] Seq=134 Ack=1673 Win=9824 Len=0 TSV=18845 TSER=2182534
    Identification: 0x1644 (5700)

No.     Time        Source                Destination           Protocol Info
     51 23.975155    192.168.75.138          192.168.75.1            TCP       54064 >
icslap [ACK] Seq=134 Ack=1807 Win=12704 Len=0 TSV=18845 TSER=2182534
    Identification: 0x1645 (5701)

No.     Time        Source                Destination           Protocol Info
     53 23.977703    192.168.75.138          192.168.75.1            TCP       54064 >
icslap [FIN, ACK] Seq=134 Ack=1808 Win=12704 Len=0 TSV=18846 TSER=2182534
    Identification: 0x1646 (5702)

No.     Time        Source                Destination           Protocol Info
     55 23.979951    192.168.75.138          192.168.75.1            TCP       54065 >
icslap [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=18847 TSER=0 WS=5
    Identification: 0x0050 (80)

No.     Time        Source                Destination           Protocol Info
     57 23.981798    192.168.75.138          192.168.75.1            TCP       54065 >
icslap [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSV=18847 TSER=2182535
    Identification: 0x0051 (81)

No.     Time        Source                Destination           Protocol Info
     58 23.984743    192.168.75.138          192.168.75.1            TCP       54065 >
icslap [PSH, ACK] Seq=1 Ack=1 Win=5856 Len=133 TSV=18848 TSER=2182535
    Identification: 0x0052 (82)
```

View at: http://buchananweb.co.uk/packet_ip_ub.txt

And in Windows it differs as starts with a random value, and then increments each TCP data segment by one each:

```
No.     Time        Source                Destination           Protocol Info
      3 0.001525    192.168.75.132         192.168.75.1           TCP      afrog > http
[SYN] Seq=0 Win=64240 Len=0 MSS=1460
    Identification: 0x008c (140)
```

```
No.     Time        Source                Destination           Protocol Info
      4 3.019628    192.168.75.132         192.168.75.1          TCP      afrog > http
[SYN] Seq=0 Win=64240 Len=0 MSS=1460
    Identification: 0x008e (142)

No.     Time        Source                Destination           Protocol Info
      7 8.968288    192.168.75.132         192.168.75.1          TCP      afrog > http
[SYN] Seq=0 Win=64240 Len=0 MSS=1460
    Identification: 0x008f (143)
…. Packets missed out …

No.     Time        Source                Destination           Protocol Info
    129 30.598774   192.168.75.132         84.53.138.18          TCP      dcutility >
http [ACK] Seq=4751 Ack=28096 Win=63188 Len=0
    Identification: 0x00d1 (209)
```

View at: http://buchananweb.co.uk/packet_ip_windows.txt

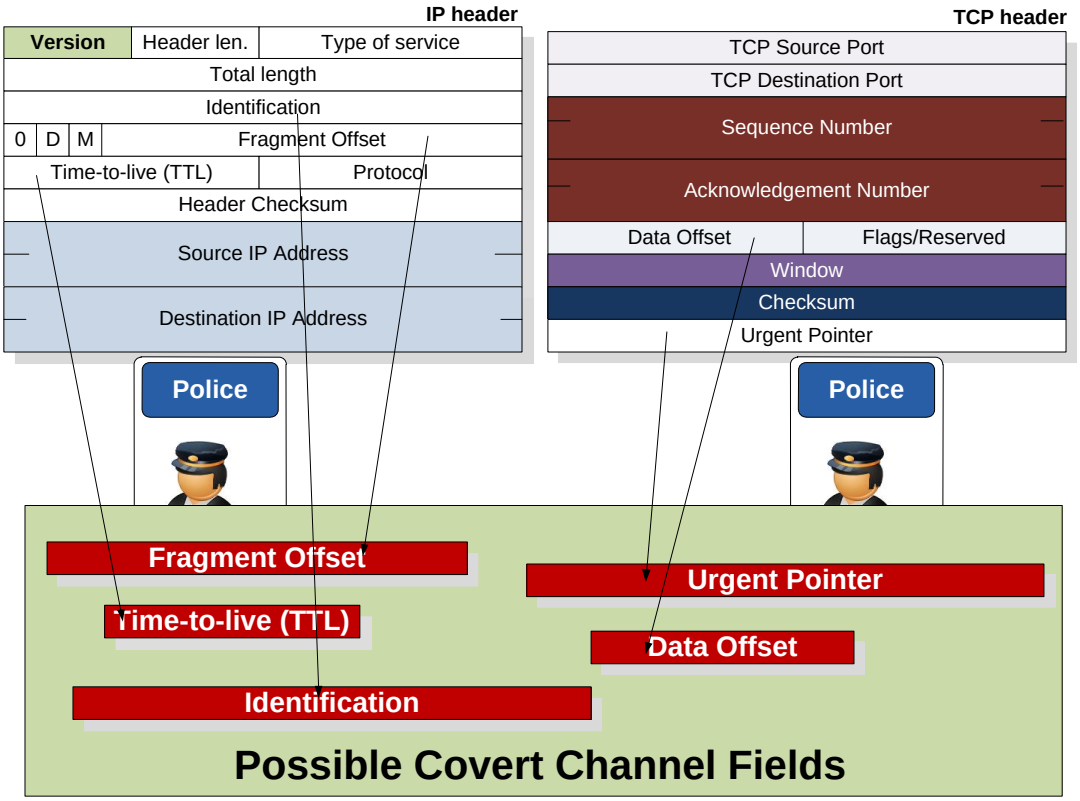


Figure 10.8 Possible fields for data hiding in IP and TCP headers

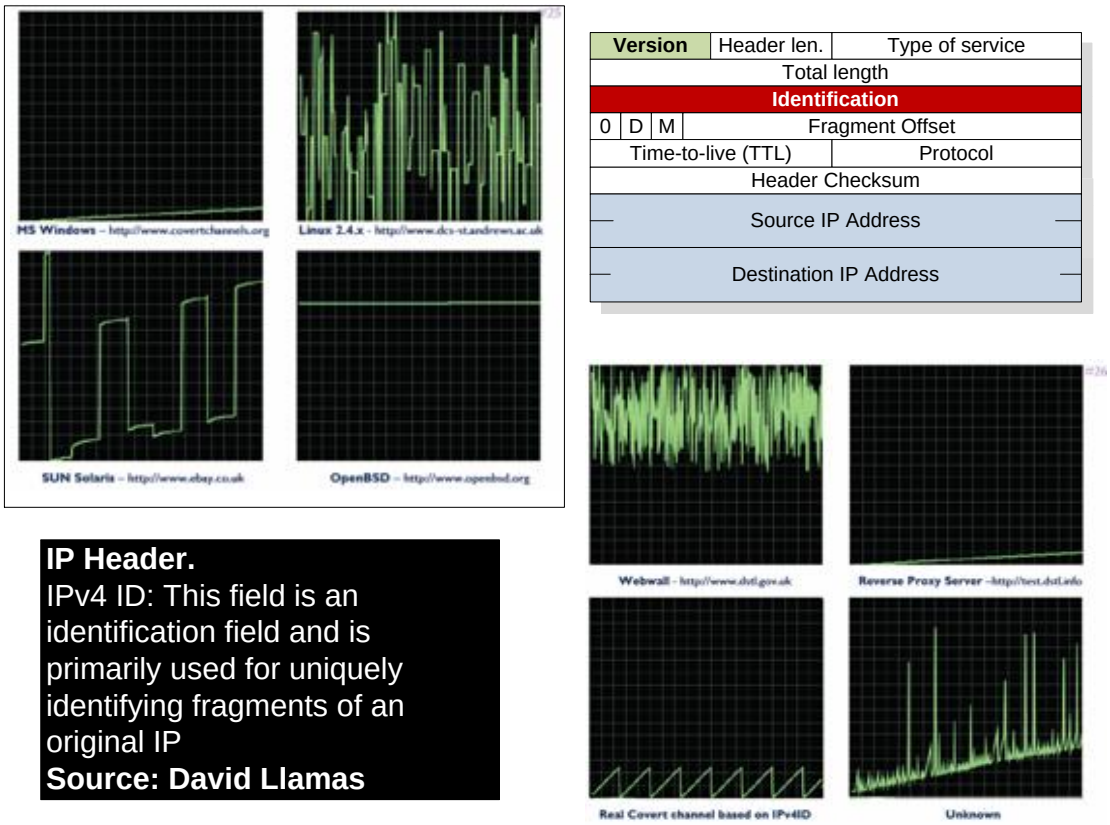


Figure 10.9 Identification field

10.6 Watermarking and Stenography

A digital watermark is either visible or invisible, and is typically a copyright mark which is added to the content. This is normally done with graphical/animation files, where an invisible element to graphics is added. For example, in Figure 8.3, the text "Bill's Graphic" has been added, but the opacity of the text has been changed from 100% down to 50%. If it was changed to 0% it would be invisible to the user, but the text would still be there. This method, though, can normally be spotted and easily deleted. Also it only works on graphics/video formats which support opacity and vector-based graphics, such as PNGs, and so on. Unfortunately bit-mapped images such as GIF and JPEG images do not support it.

There are literately an endless number of ways that stenography can be used. One example, is to add information into files which can not actually used, such as in images files. Figure 8.5 shows an example where a GIF file contains a colour table, of which, typically, not all the colours are used in any image. Thus text can be added to the file, which will never actually be seen.

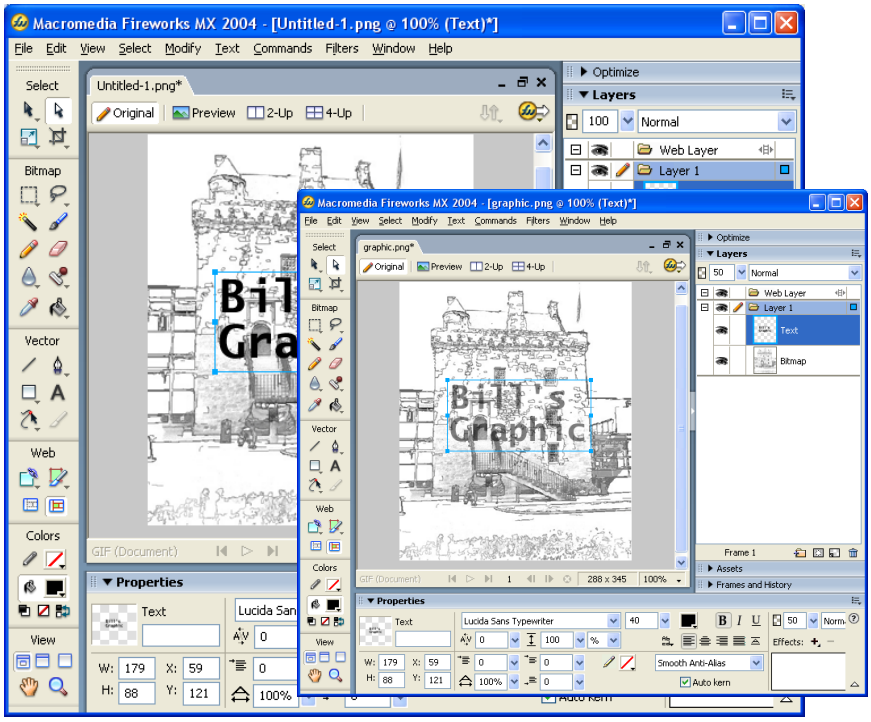


Figure 10.10 Information hiding classifications

Another way is to add information to images, which have no visual effect on the image. This is typically to high information in the high-frequency changes in images, such as in Figure 8.4.

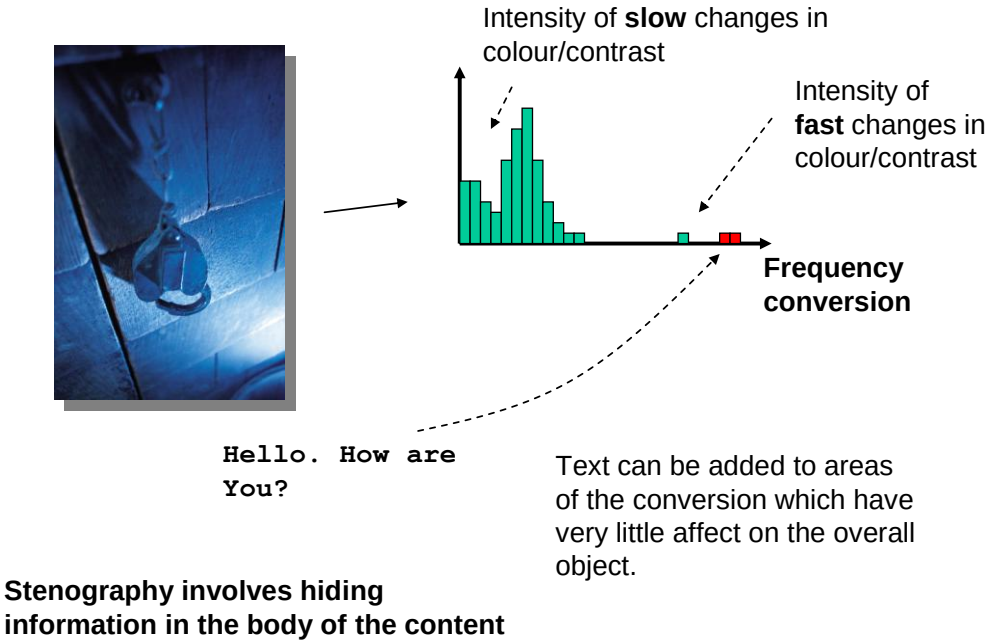


Figure 10.11 Information hiding classifications

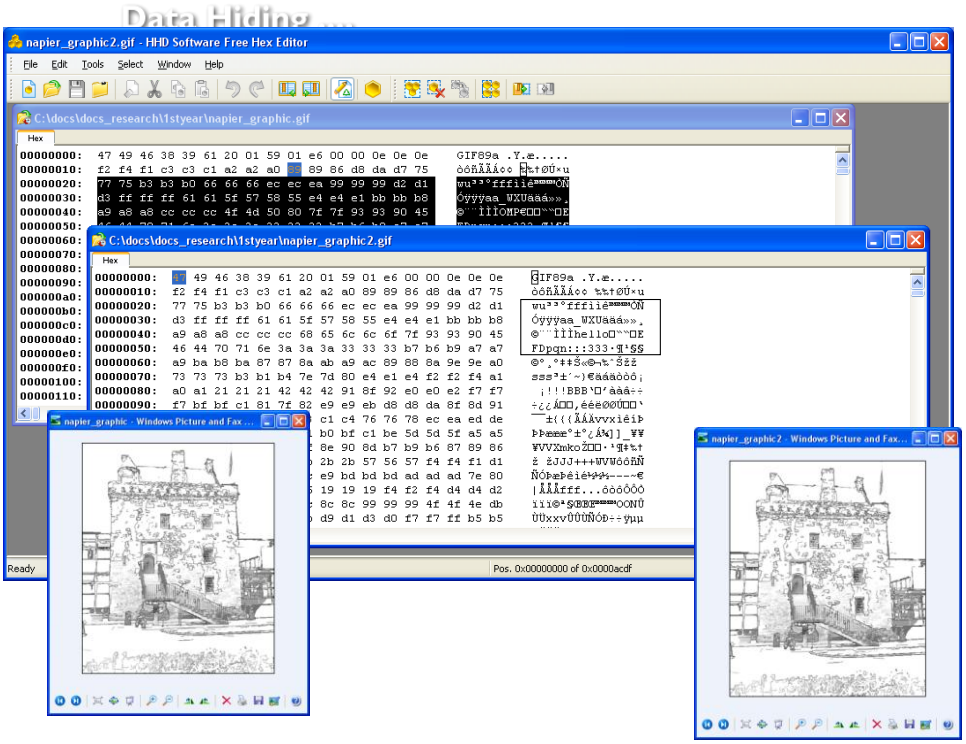


Figure 10.12 Information hiding classifications

10.7 Hiding File contents

Most computer file names are made up of a filename and a file extension, where the extension is used to define the classification of a file, such as for a word processing document, a spreadsheet, and so on. Graphics files, for example, are often used in investigations, thus it is important to identify them on a file system. One method is to search for the file extension, such as JPEG, GIF or PNG through the current folder or subfolders, such as with:

```
dir *.jpg /s
```

from the command prompt, whereas another method is to use the find utility in Windows. A search of the key file formats might include:

Microsoft Word documents	.DOC or .RTF
Image files	.GIF, .JPG, .PNG
Presentation files	.PPT
Spreadsheets	.XLS

One problem with the method of searching for files by their file extension is when the files have been obfuscated in some way, such as where the file extension of the file has been changed, or where the images have been embedded within other documents.

File contents

Files contain binary information which are typically read one byte at a time. In order the make the binary information readable the binary digits are typically interpreted in a hexadecimal format as it is relatively easy to convert from binary to hexadecimal, and vice-versa. With text files, the characters are typically stored as ASCII characters, which can be read directly in a readable format. For example a ZIP file has the following bit sequence at the start of the file:

```
0101 0000 0100 1011 0000 0011 0000 0100
```

which is difficult to remember or to define, thus the hexadecimal equivalent of:

```
50 4B 03 04h
```

is easier to define. Some of the 8-bit binary values will produce a printable character, such as the values from 20h to 7Eh. For example 20h is a space character, 21h is '!', and so on. Thus the hexadecimal value of:

```
50 4B 03 04h
```

when interpreted as ASCII characters is displayed as:

```
PK▫▫
```

where ▫ is a non-printing character. Thus binary files can contain some information which can be interpreted by a viewer which displays each byte as an ASCII character. Unfortunately ASCII is a rather limited character set, and does not support enhanced characters, such as for mathematical symbols. Thus other character sets can be used to save information. A good example is Unicode which extends the character sets with more bits, typically 16-bits for each character. Thus for files stored as 16-bit Unicode, the characters must be interpreted 16 bits at a time. For example, in Figure 9.1 a PowerPoint file has been created and an image of pics_cookie_transparent _32colors.gif has been imported into the file. It can be seen in Figure 9.2 that the original name of the file is stored as:

```
00 70 00 69 00 63 00 73 ...
```

which is interpreted as:

```
▫p▫i▫c▫s▫_▫c▫o▫o▫k▫i▫e▫_▫t▫r▫a▫n▫s▫p▫a▫r▫e▫n▫t▫_▫3▫2▫c▫o▫l▫o▫r▫s
```

thus the lower 8 bits of the 16-bit character still displayed as an ASCII character, but a search for this name, for example, must have to involve searching 16 bit values, at a time. The similarity between ASCII and Unicode can be seen from:

Char	**ASCII (hex)**	**Unicode**
'A'	41h	0041h

‘B’	42h	0042h
‘C’	43h	0043h

and so on.

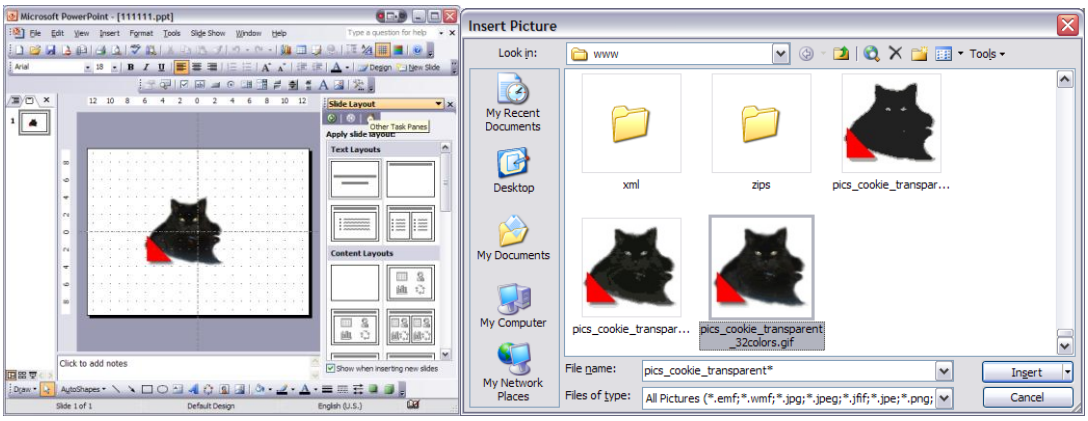


Figure 10.13 PowerPoint example

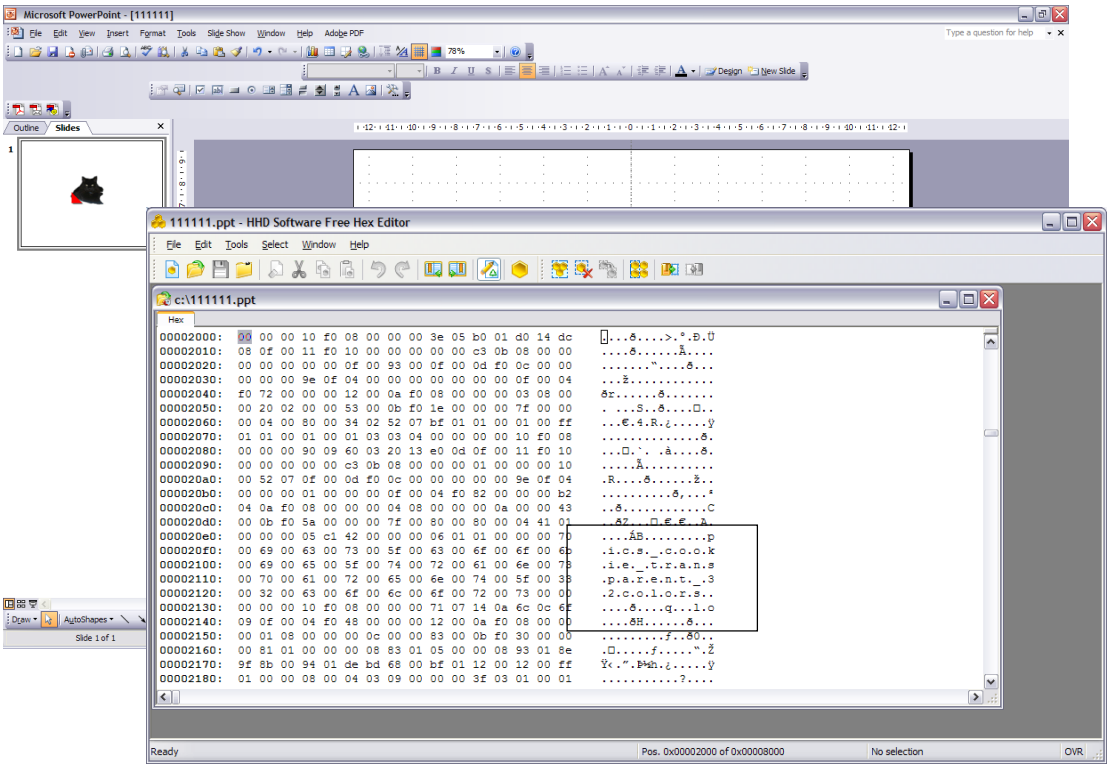


Figure 10.14 PowerPoint file format

The pattern stored is thus in the form:

```
00 63 00 6f 00 6f 00 6b 00 69 00 65 00 5f 00 74     .c.o.o.k.i.e._.t
00 72 00 61 00 6e 00 73 00 70 00 61 00 72 00 65     .r.a.n.s.p.a.r.e
```

The search string can be modified so that it looks for the string of "\0c\0o\0o\0k\0i\0e" rather than for “cookie”. The following code snippet achieves this:

```
using System;
using System.IO;

namespace ConsoleApplication1
{
   class Class1
   {

      static void Main(string[] args)
      {
         DirectoryInfo di = new DirectoryInfo("c:\\test123");
         FileInfo[] rgFiles = di.GetFiles("*.*");
         foreach(FileInfo fi in rgFiles)
         {
            StreamReader f = new StreamReader("c:\\test123\\"+fi.Name);
            string s = f.ReadToEnd();
            string search = "\0c\0o\0o\0k\0i\0e";
            if (s.LastIndexOf(search)>0)
            {
                  Console.WriteLine("Search signature found, name: " + fi.Name);

            }
         }
         Console.WriteLine("Press return to end..");
         Console.ReadLine();
      }
   }
}
```

Another method is to create a byte array with the byte sequence to search for, and then convert it to a string, such as with:

```
         byte [] b = {0,(byte)'c',0,(byte)'o',0,(byte)'o',0,
                                    (byte)'k',0,(byte)'i',0,(byte)'e'};
         System.Text.ASCIIEncoding enc = new System.Text.ASCIIEncoding();
         string search  = enc.GetString(b);
```

The standard Windows search does not cope well with binary searches, but the standard find utility copes better, such as:

```
C:\test123>find /?
Searches for a text string in a file or files.

FIND [/V] [/C] [/N] [/I] [/OFF[LINE]] "string" [[drive:][path]filename[ ...]]

  /V         Displays all lines NOT containing the specified string.
  /C         Displays only the count of lines containing the string.
  /N         Displays line numbers with the displayed lines.
  /I         Ignores the case of characters when searching for the string.
  /OFF[LINE] Do not skip files with offline attribute set.
  "string"   Specifies the text string to find.
  [drive:][path]filename
             Specifies a file or files to search.

If a path is not specified, FIND searches the text typed at the prompt
or piped from another command.

C:\test123>find "GIF89a" *.*

---------- 111111.PPT
---------- 123.JPG
---------- AA.GIF
GIF89aä♥3☻µ
---------- AGENTS02.GIF
GIF89a╥
--------- AGENT_GRAPHIC01.GIF
GIF89a▌
---------- AGENT_GRAPHIC02.GIF
GIF89a(☻b
```

```
---------- FLASH_NETWORK_EMULATORS2.JPG
---------- PRES01.PPT
---------- SRCCODE.ZIP
```

10.7.1 GIF files

The graphics interchange format (GIF) is the copyright of CompuServe Incorporated. Its popularity has increased mainly because of its wide usage on the Internet. Most graphics software support the Version 87a or 89a format (the 89a format is an update the 87a format). Both the basic specification of:

- A header with GIF identification.
- A logical screen descriptor block which defines the size, aspect ratio and color depth of the image place.
- A global color table. Color tables store the color information of part of an image (a local color table) or they can be global (a global table).
- Data blocks with bitmapped images and the possibility of text overlay.
- Multiple images, with image sequencing or interlacing. This process is defined in a graphic-rendering block.
- Compressed bitmapped images.

Blocks can be specified into three groups: control, graphic-rendering and special purpose. Control blocks contain information used to control the process of the data stream or information used in setting hardware parameters. They include:

- GIF Header – which contains basic information on the GIF file, such as the version number and the GIF file signature.
- Logical screen descriptor – which contains information about the active screen display, such as screen width and height, and the aspect ratio.
- Global color table – which contains up to 256 colors from a palette of 16.7M colors (i.e. 256 colors with 24-bit color information).
- Data subblocks – which contain the compressed image data.
- Image description – which contains, possibly, a local color table and defines the image width and height, and its top left coordinate.
- Local color table – an optional block which contains local color information for an image as with the global color table, it has a maximum of 256 colors from a palette of 16.7M.
- Table-based image data – which contains compressed image data.
- Graphic control extension – an optional block which has extra graphic-rendering information, such as timing information and transparency.
- Comment extension – an optional block which contains comments ignored by the decoder.
- Plain text extension – an optional block which contains textual data.
- Application extension – which contains application-specific data. This block can be used by a software package to add extra information to the file.
- Trailer – which defines the end of a block of data.

The key to identifying the GIF file is the six bytes long initial header which identifies the GIF signature and the version number of the chosen GIF specification. Its format is (Figure 9.3):

- 3 bytes with the characters ‘G’, ‘I’ and ‘F’.
- 3 bytes with the version number (such as 87a or 89a). Version numbers are ordered with two digits for the year, followed by a letter (‘a’, ‘b’, and so on).

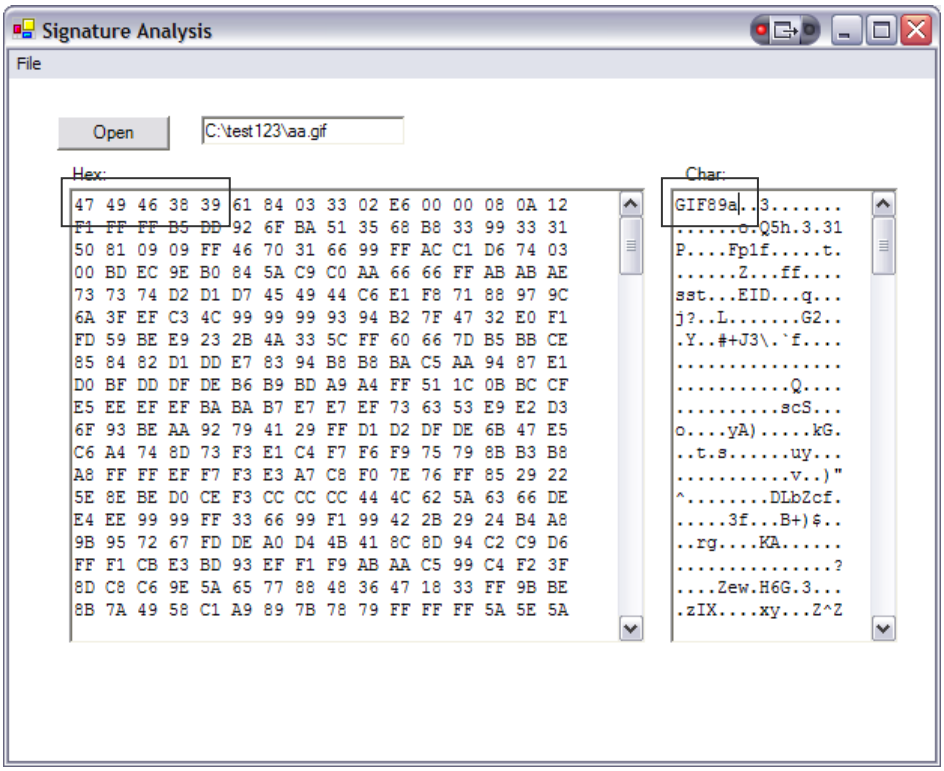


Figure 9.3: GIF file header

10.7.2JPEG file format

JPEG is a standard compression technique. The files which contain JPEG images normally complies with JFIF (JPEG file interchange format) which is a defined standard file format for storing a gray scale or color image. The data within the JFIF contains segments separated by a 2-byte marker. This marker has a binary value of 1111 1111 (FFh) followed by a defined marker field. If a 1111 1111 (FFh) bit field occurs anywhere within the file (and it isn’t a marker), the stuffed 0000 0000 (00h) byte is inserted after it so that it cannot be read as a false marker. The uncompression program must then discard the stuffed 00h byte.

Some of the key markers are:

- Start of image (FFD8h). The segments can be organized in any order but the start-of-image marker is normally the first 2 bytes of the file. Refer to Figure 9.4 and 10.5 for the file format.
- Application-specific type 0 (FFE0h). The JFIF header is placed after this marker.

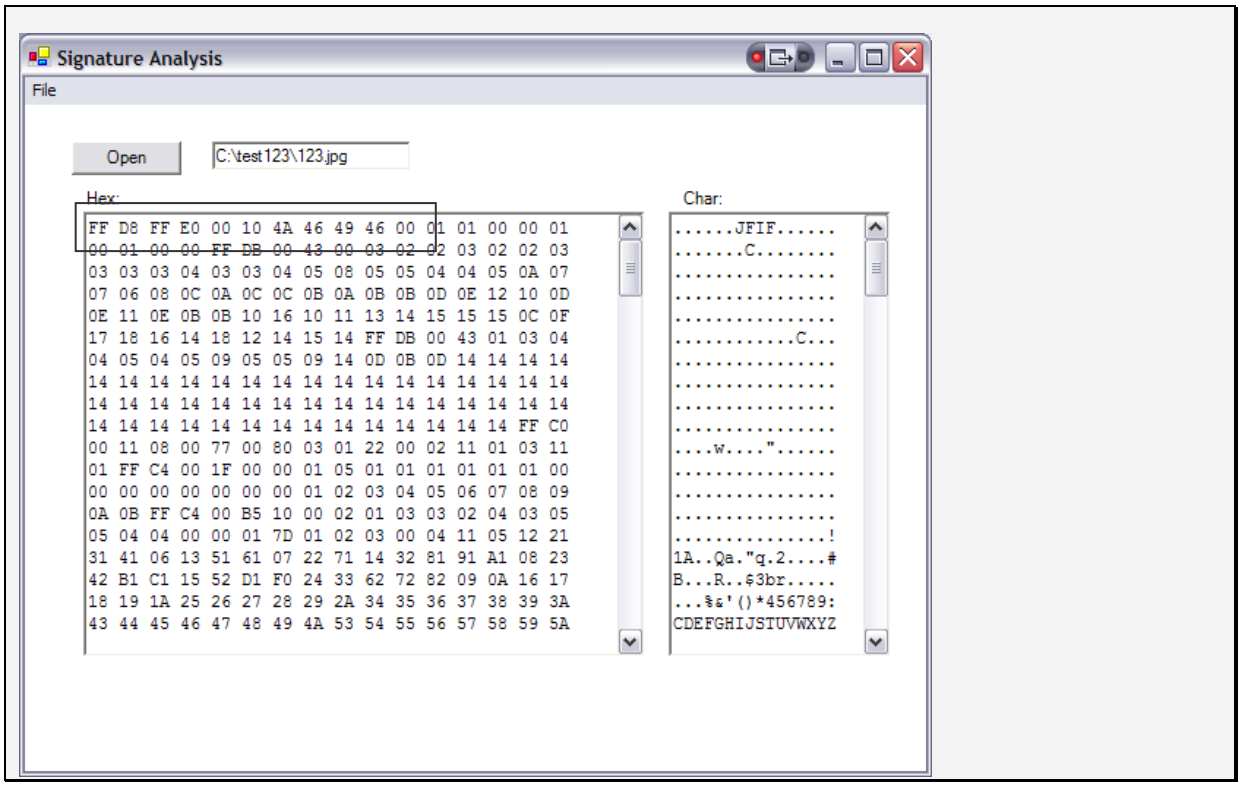


Figure 10.15 JPEG file reading

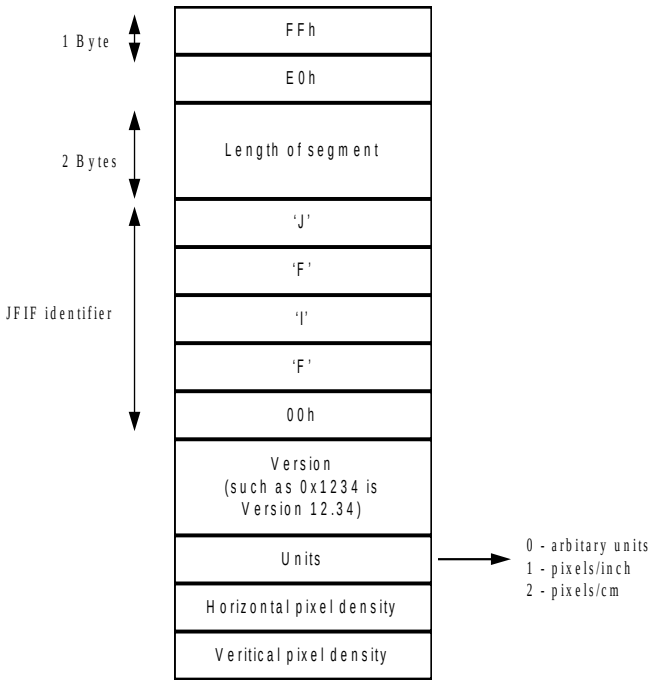
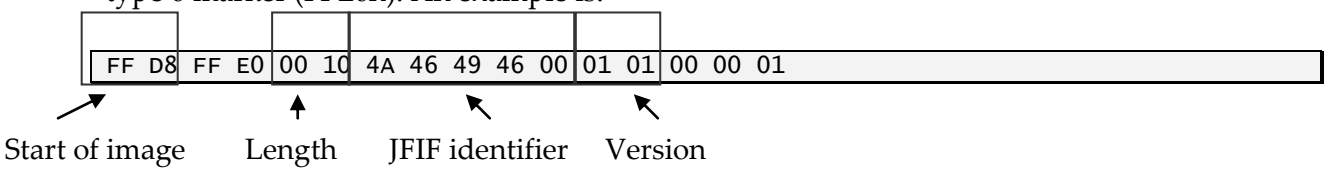


Figure 10.16 JFIF header information

JPEG graphics files have a JFIF header which begins with the application-specific type 0 marker (FFE0h). An example is:



The first 11 bytes can thus be used to identify the start of a JPEG image file, where the hex value of a 'J' is 4Ah, an 'F' is 46h, an 'I' is 49h. Thus the string "JFIF" is represented with the hexadecimal pattern of 4A464946h (Figure 9.4).

10.7.3ZIP file format

From a forensics point-of-view the detection of a ZIP file can often mean that a file(s) has/have been compressed into a single file. The basic format of its header is:

Byte pos.	Name	No. of bytes	Contents	Description
00	ZIPLOCSIG	4	50 4B 03 04h	File signature
04	ZIPVER	2		Version required for extraction
06	ZIPGENFLG	2		General purpose bit flag
08	ZIPMTHD	2		Compression method
0A	ZIPTIME	2		Time last modified
0C	ZIPDATE	2		Date last modified
0E	ZIPCRC	4		CRC-32
12	ZIPSIZE	4		Compressed size
16	ZIPUNCMP	4		Uncompressed size
1A	ZIPFNLN	2		Filename length
1C	ZIPXTRALN	2		Extra field length
1E	ZIPNAME			Filename

The ZIP file format contains quite an amount of data about the contents of the ZIP file. Apart from the signature, it can be seen from Figure 9.6 that the file names of the files contained within the ZIP file are also contained in the header.

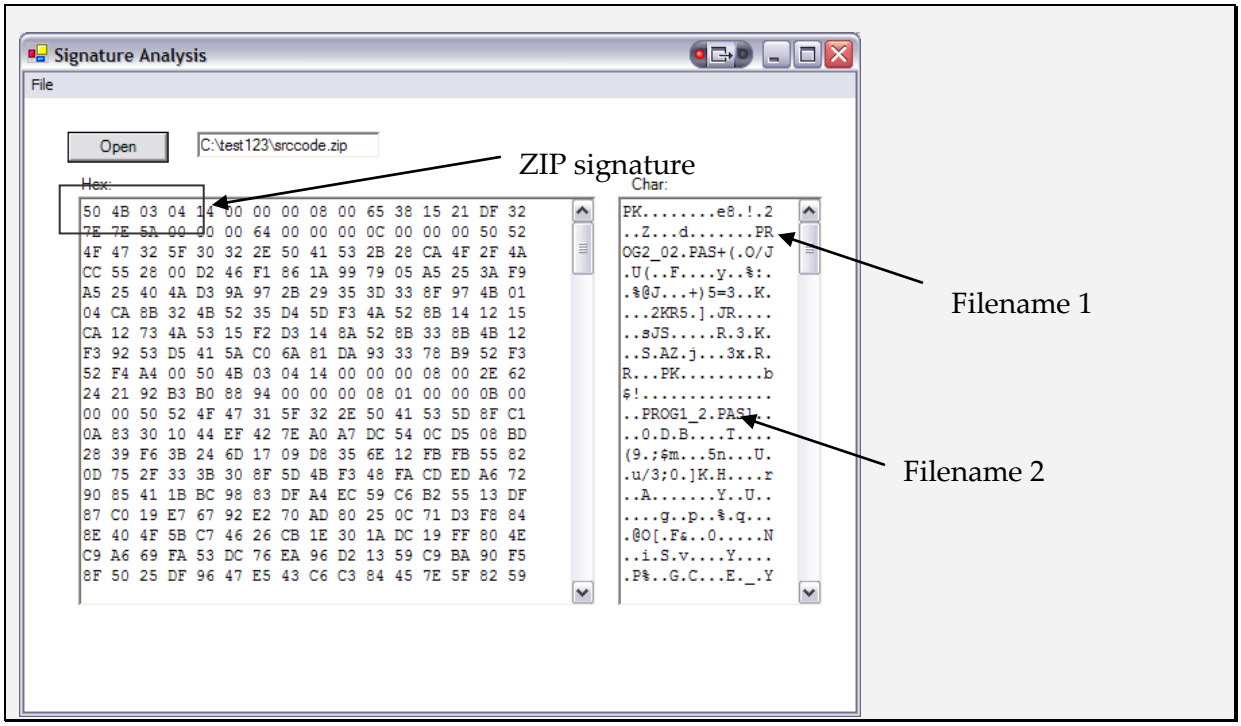


Figure 10.17 ZIP file information

10.8 References

Berg, S. (1998). Glossary of Computer Security Terms, http://packetstormsecurity.org/docs/rainbow-books/NCSC-TG-004.txt.

Gligor, V. D. (1993). A Guide to understanding Covert Channel Analysis of Trusted Systems. Technical Report NCSC-TG-030, National Computer Security Centre.

Kwecka Z, (2006), Hons Project Dissertation, Award Winner, Young Software Engineer of the Year Award. http://www.dcs.napier.ac.uk/~bill/zk.pdf.

Lampson, W. (1973). "A note on the Confinement Problem.Communications of the ACM." (16(10)): 613-615.

Lipner, S. B. (1975). "A note on the Confinement Problem." Operating Systems Review, 9(5): 192-196.

Llamas D, (2004), Hons Project Dissertation, Award Winner, Young Software Engineer of the Year Award. http://www.dcs.napier.ac.uk/~bill/PROJECTS/2004/ david_llamas.pdf.

Proctor, N. E. and P. G. Neumann (1992). Architectural implications of Covert Channels.15th National Computer Security Conference, 28-43.

Route (1996). "Project Loki: ICMP Tunnelling." Phrack Magazine 7(49).

Rowland, C. H. (1996). Covert Channels in the TCP/IP Protocol Suite, http://www.firstmonday.dk/issues/issue2_5/rowland/.

10.9 Tutorial

The main tutorial is at:

On-line tutorial: http://buchananweb.co.uk/adv04.html

Download and install: http://buchananweb.co.uk/dotnetclientserver.zip

10.10 Exercises

10.1 Select [Encryption->Hashing tab] Determine the Base-64 hash signature for "test" for the following:

MD5:

SHA-1:

SHA-256

How many bits does each of these signatures have:

10.2 Select [Encryption->Hash (Collision) tab] Determine the ASCII message for the following hash signatures:

```
AD5F82E879A9C5D6B5B442EB37E50551

15B6AF8D85CBE1229C7150E10D5A55BD3417B40C

EEBC8CF2B3B360C51A34E0E8EBD98B8F37F348B7

1F7BA58706F9D405023DA32864D059C8
```

10.3 **Select [Encryption->Base-64 tab]** Determine the ASCII message from the following Base-64 messages:

```
SGVsbG8gaG93IGFyZSB5b3U/

Q2FuIHlvdSByZXZlcnNlIGl0Pw==

VGhpcyBpcyBhIHNhbXBsZSBwaWVjZSBvZiB0ZXh0Li4u
```

10.4 **Select [Encryption->Base-64 tab]** Determine the Base-64 string for the following encrypted strings in 3DES and AES, which have been encrypted with the key word of "sample1234":

```
napier

fullstop

apple.tree
```

How many bits does the result have, and how does it vary for the following words, and explain the reason for the changes it the output size:

```
aaaaa
aaaaaa
aaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaaa
aaaaaaaaaaa
```

What does the "=" represent at the end of the encrypted string?

10.5 **Select [Encryption->Private-key encryption tab]** The result of an encryption process is "7xCJIB1RVG5/2HQFrDH9Kw==", which was encrpyed from either "foxtrot", "orangepeel", or "interrupt".

Which password was used, what encryption type was used, and what was the original message:

10.6 Select [Encryption->Brute force tab] Using a brute-force directionary search, determine the AES encryption key for the following:

Determine the encryption key, and the orginal message:

2AC3B3211DEADC97C824307090BD33EA

194E22BF7A463D8A048140400497DCA7

F2BE257B9B13B72634013D9E528B6A9F

60FA30C4E4EAFF88EB741BCEEE976CD7D66DC12EBE2C9425C331F4B01FC65A2A

10.7 Select [Encryption->Public-key encryption/decryption tabs] Download the following public-key:

http://buchananweb.co.uk/publickey01.txt

and use it to encrypt the word "test", and prove the result is:

"17500DDDBD378…

10.8 Select [Encryption->Public-key encryption/decryption tabs] Download the following private-key:

http://buchananweb.co.uk/privatekey01.txt

and provide that it can decrypt the cipertext.

10.9 Select [Encryption->Public-key encryption/decryption tabs] Using the private-key (http://buchananweb.co.uk/privatekey02.txt), and the following cipher stream (copy it from the PDF document), determine the message:

2FB7C6F9719A05E79FA0591E92CE1884DB9CDB015F4F29D405B7ED5216
03AFEB404E9884BE0F83597C3054BC721CD0F15E39091B7894B11929CA
CFE7B77F7A29DD41ED3AC27D4C825157B61A1775B104045731A1B3CDD8
BDDCB091544D2FAC7D50DEBC8AD79D1BE1F73999D7FE6B8E8AB61142B7
1A0F274E0053D9C1FE3B80F3

What is the message:

10.10 Select [Encryption->Digital certificate tab] Open up fred.cer, and determine its main parameters:

Certificate details:

10.11 Select [Encryption->Digital certificate tab] Open up sample01, sample02, sample03, sample04 and sample05, and determine their passwords:

Passwords on certificates:

10.12 Select [Coding->Ex-OR tab] If the message is "Testing", what is the single digital Ex-OR key for the following Base-64 strings:

```
NwYQFwoNBA==

EiM1Mi8oIQ==

Lh8JDhMUHQ==
```

10.13 Select [Coding->Encoding tab] Determine the message for the follow encoding formats:

```
48656C6C6F20686F772061726520796F753F

2431323334353637383924

VGVzdGluZyAiMTIzIiAuLi4=
```

10.14 Select [Coding->Caesar code tab] Determine the message for the following Caesar codes:

```
OLSSV OVD HYL FVB


MABL BL HGER T FXLLTZX

PEEAT RDGT
```

10.15 **Select [Binary Reader],** for open the first file (file1). The output should be something like in Figure 1.

Refer to the Appendix given, and determine the format of the file.

What is the format of the file (such as GIF, JPEG, ZIP, etc):

Now repeat for files 2 to 10, and complete the following table:

Name	*File format (circle correct one)*	*Is there any copyright information in the file (or associated information that is readable)?*
File2	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	
File3	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	
File4	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	
File5	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	
File6	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	
File7	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	
File8	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	
File9	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	

File10	*DOC/PPT/XLS/JPEG/GIF/WMF/ZIP*	

10.16 **Select [Binary Reader],** for the ZIP file:

Identify the file name contained within the ZIP file:

What is the termination character used to terminate the file name:

Can you tell the date and time that it was last modified?

10.17 For other binary file formats, determine their signature (if possible).

PDF file signature:

SWF (Flash) file signature:

DLL file signature:

RTF file signature (open up a Word document, and save it in an RTF file format):

10.18 **Select [Coding],** performance a frequence analysis on the following, and determine the original text:

```
XQG XP MJG PAEDM XBBKEEGQBGD XP BXC-LKMGE MGBJQXHXFT XBBKEEGO AQ MJG KDY AQ
MJG 1880D. AM VYD OKG MX MJG YCGEABYQ BXQDMAMKMAXQ OGCYQOAQF MJYM Y DKERGT
UG KQOGEMYIGQ GRGET 10 TGYED. YD MJG LXLKHYMAXQ AQ MJG KDY AQBEGYDGO, AM
MXXI YQ AQBEGYDAQF YCXKQM XP MACG MX LEXOKBG MJG DMYMAD-MABD. UT MJG 1880D,
AM HXXIGO HAIGHT MJYM MJG 1880 DKERGT VXKHO QXM UG BXCLHGMG KQMAH 1890. MX
XRGEBXCG MJAD, JGECYQ JXHHGEAMJ (VJX VXEIGO PXE MJG FXRGEQCGQM) OGRADGO Y
CY-BJAQG MJYM YBBGLMGO LKQBJ BYEOD VAMJ AQPXECYMAXQ XQ MJGC. MJGDG BYEOD
YHHXVGO Y BKEEGQM MX LYDD MJEXKFJ XQHT VJGQ MJGEG VYD Y JXHG LEGDGQM.

JXHHGEAMJ'D GHGBMEXCGBJYQABYH CYBJAQG VYD GZMEGCGHT DKBBGDDPKH YQO VYD KDGO
AQ MJG 1890 YQO 1900 BGQDKDGD. JG GRGQ PXKQOGO MJG BXCLYQT MJYM VXKHO HYMGE
UGBXCG AQMGEQY-MAXQYH UKDAQGDD CYBJAQGD (AUC).
```

10.19 **Select [Coding],** performance a frequence analysis on the following, and determine the original text:

```
FN 1985, GLLBK TGH IGOFNE AFXXFUMBJ JFSKH. JIK HGBKH CX JIK SGUFNJCHI TKWK
NCJ GH EWKGJ GH KRLKUJKA, GNA JIK GLLBK FF TGH XGUFNE G EWKGJ AKGB CX
UCSLKJFJFCN XWCS CJIKW SGNMXGUJMWKWH. SGNY LKCLBK GJ JIK JFSK, FNUBMAFNE
QFBB EGJKH, TKWK GAOFHFNE GLLBK JC CLKN-ML JIK SGWDKJ XCW SGUFNJCHI
```

UCSLMJKWH QY GBBCTFNE CJIKW SGNMXGUJMW-KWH QMFBA JIKFW CTN HYHJKSH, MNAKW HJWFUJ BFUKNHK GWWGNEKSKNJH. QFBB EGJKH IGA GAOFHKA JIKS JIGJ JIKY HICMBA JFK ML TFJI UCSLGNFKH HMUI GH IL GNA GJ&J. ICTKOKW, GLLBK IKBA CNJC QCJI JIKFW SGU CLKWGJFNE HYHJKS, GNA JIKFW IGWATGWK, TIFUI JIKY QKBFKOKA TKWK JCJGBBY FNJKWJTFNKA. G SGU UCMBA NCJ KRFHJ TFJICMJ QCJI FJH CLKWGJFNE HYHJKS GNA FJH IGWATGWK. WGJIKW JIGN CLKN JIK SGWDKJ ML, GLLBK AKUFAKA JC JWGSLBK UBCNKWH, KHLK-UFGBBY FN HCXJTGWK UBCNKWH. GLLBK'H XFWHJ JGWEKJ TGH AFEFJGB WKHKGWUI, TIC IGA AKOKBCLKA EKS XCW JIK LU. AFEFJGB WKHKGWUI QKBFKOKA JIGJ JIKY IGA QCWWCTKA JIK BCCD-GNA-XKKB CX JIK SGU CLKWGJFNE HYHJKS, QMJ NCJ JIK GUJMGB JKUINCBCEY. GLLBK FSSKAFGJKBY HICJ EKS CMJ CX JIK TGJKW TIKN GLLBK'H BGTYKWH, FN 1985, OFHFJKA AFEFJGB WKHKGWUI GNA JIWKGJKNKA JIKS TFJI UCMWJ GUJFCN. GJ JIK JFSK, FQS IGA QKKN DKKN JC BFUKNHK EKS XCW JIKFW CTN LWCAMUJH, QMJ JIKY TKWK XWFEIJKNKA GTGY COKW JIK XKGW CX BFJFEGJFCN, GNA JIGJ TGH JIK KNA CX EKS.

GLLBK JIKN JMWNKA JC SFUWCHCXJ JC IKGA CXX JIKFW GJJKSLJ GJ LWCAMUFNE G EMF. QFBB EGJKH, JICMEI, IGA SMUI EWKGJKW HJWKNEJI JIGN AFEFJGB WKHKGWUI GEGFNHJ GLLBK. IFH SGFN LCFNJ TGH JIGJ JIK JWMK CWFEFNGJCW CX JIK EMF TGH RKWCR. JIMH, XCW FJH SFUWCHCXJ TFNACTH, FJ TGH RKWCR'H FAKGH JIGJ TKWK QKFNE MHKA, GNA NCJ GLLBK'H. QFBB EGJKH, JICMEI, IGA GNCJIKW JWMSL UGWA: FX GLLBK TKWK ECFNE JC HJCL SFUWCHCXJ XWCS LWCAMUFNE TFNACTH JIKN SFUWCHCXJ TCMBA HJCL LWCAMUFNE GLLBFUGJFCN HCXJTGWK XCW JIK SGUFNJCHI. GLLBK DNKT JIGJ JIKY NKKA-KA SFUWCHCXJ SCWK JIGN SFUWCHCXJ NKKAKA GLLBK. FN JIK XGUK CX G BGUD CX FNOKHJSKNJ FN JIKFW GLLBFUGJFCN HCXJTGWK, GLLBK HFENKA G UCNJGUJ TFJI SFUWCHCXJ TIFUI HJGJKA JIGJ SFUWCHCXJ TCMBA:

'IGOK G NCN-KRUBMHFOK, TCWBATFAK, WCYGBJY-XWKK, LKWLKJMGB, NCNJWGNHXKWGQBK BFUKNHK JC MHK AKWFOGJK TCWDH FN LWKHKNJ GNA XMJMWK HCXJTGWK LWCEWGSH, GNA JC BFUKNHK JIKS JC GNA JIWCMEI JIFWA LGWJFKH XCW MHK FN JIKFW HCXJTGWK LWCEWGSH'

TIFUI QGHFUGBBY EGOK SFUWCHCXJ UGWJK QBGNUIK XCW GBB XM-JMWK OKWHFCNH CX JIKFW HCXJTGWK, GNA TKWK PMFJK XWKK JC QCWWCT TIFUI KOKW XKGJMWKH JIKY TGNJKA. ZCIN HUMBBY GJ GLLBK HFENKA FJ, GNA EGOK GTGY CNK CX JIK SCHJ BMUWGJFOK SGWDKJH FN IFHJCWY. QGHFUGBBY, GLLBK TGH QMYFNE LKGUK TFJI SFUWCHCXJ, QMJ FJ TGH LKGUK TFJI G BCNE-JKWS UCHJ.