Cryptographic and Cryptanalytic Methods Used in Computer Viruses and Anti–Virus Software

Dr. Vesselin Bontchev, anti–virus researcher
FRISK Software International
Thverholt 18, IS-105 Reykjavik, Iceland
E–mail: bontchev@complex.is

***Abstract**: The paper describes the various cryptographic and cryptanalytic techniques, which the author has observed as being used in computer viruses and anti–virus software. Most of what the anti–virus people call "encryption" is not really encryption but encoding. The paper emphasizes this difference in terminology and describes in detail only the techniques that can correctly be classified as "cryptographic" and "cryptanalytic" from a cryptographer's point of view.*

1. Cryptographic and Cryptanalytic Methods Used in Computer Viruses

In the first part of this paper, we shall examine the various cryptographic and cryptanalytic techniques that are used by various computer viruses.

1.1 Cryptographic Methods Used in Computer Viruses

In the first section, we shall describe the various cryptographic methods used by the virus writers in their creations.

1.1.1 Encoding

Originally, the anti–virus programs were looking for viruses by scanning the executable files for short sequences of bytes extracted from the virus body. These sequences were chosen so that they were present in every replicant of the virus, while at the same time they were unlikely to be found in normal, uninfected programs. Such sequences are known under the name *scan strings* – although many people still use the inaccurate term "signatures" for them. (That term is inaccurate for two reasons. *First*, it is misleading, since it has nothing to do with digital signatures. *Second*, it implies that just like every human has a single signature unique to them, every virus has a single sequence of bytes typical for it – while, in reality, it is possible to select many different scan strings for the one and the same virus.)

Naturally, the virus writers sought to defeat this approach. The obvious way to do it was to make sure that it was not possible to select a sufficiently long sequence of bytes from the virus body – a sequence, which could be found in all replicants of the virus.

There are two possible ways to achieve this goal. The easiest one is to encrypt the virus body with a key that changes every time the virus replicates. The more difficult one is to rewrite the whole body of the virus every time it replicates, using the fact that one and the same program (algorithm) can be implemented in a practically infinite number of ways. Naturally, the vast ma-

jority of the virus writers chose the easiest approach – there are only very few viruses that use the more difficult one.

Of course, the encrypted body of the virus must be decrypted, before it can execute. For this purpose, such viruses usually contain a small unencrypted part – a part that receives control first and decrypts the rest of the virus body before transferring control to it. Since it *has* to know how to decrypt it, this part must contain both the decryption algorithm and the decryption key. In addition, since they are both present together with the encrypted part of the virus, from a cryptographer's point of view this is not really encryption – it is more appropriate to call it *encoding*. Still, viruses that use this approach are known among the anti–virus researchers as "*encrypted*".

A typical example of such a virus (and also one of the first viruses of this kind) was the `Cascade.1701.A` virus. It uses the following algorithm to "decrypt" the unencoded part of its body:

```
void Decrypt (unsigned char *body,
              unsigned short int filesize)
{
     unsigned short int i, key1, key2;

     key1 = filesize + BODYSIZE;
     key2 = BODYSIZE - 1;
     for (i = 0; key2; i--)
     {
          *((unsigned short int *) (body + i)) ^= key1++;
          *((unsigned short int *) (body + i)) ^= key2--;
     }
}
```

where `filesize` is the size of the original (uninfected) file that the virus has attached itself to and `BODYSIZE` is the length of the encoded part of the virus body (which is a constant for all replicants of the virus). In other words, they encoded part of the virus body is encoded with a key which depends on the size of the infected file.

Although the above algorithm probably looks ridiculously trivial from the point of view of a professional cryptographer, it is one of the most complex "encryption" algorithms used by computer viruses for the purposes of evading the anti–virus programs. Most other viruses use even more trivial algorithms – like a single XOR or ADD with a variable key.

The natural reaction of the anti–virus producers when faced with such viruses was to start selecting their scan strings from the part of the virus that remained always one and the same across replicants – namely, the decryptor. The virus writers attacked this approach by re-creating the decryptor every time the virus replicated and using the fact, that one and the same (trivial) decryption algorithm could be implemented in a huge number of different ways. Computer viruses that use this approach are known as *polymorphic*. Finally, viruses that do not encode a large part of its body but, instead, rewrite the whole of it each time they replicate are known as *body-polymorphic* or *metamorphic*.

Some viruses carry the "encryption" approach to great lengths. For instance, the `Dark-Paranoid` virus keeps its body encrypted even at runtime ([Kaspersky98]). Only a single in-

struction – the one that is currently being executed – is decrypted and then it is encrypted back immediately after execution. The virus achieves this by intercepting `INT1` – the so-called "debug interrupt". It has been normally designed for debugging purposes and the program that has intercepted it receives control every time before an instruction is about to be executed. This allows debuggers to "single-step" through the code they are debugging. The `DarkParanoid` virus uses this feature to decrypt the next instruction of its body that has to be executed and to re-encrypt the one that has just been executed. The part of the virus that intercepts `INT1` (the `INT1` "handler") has, of course, to be unencrypted – but it is polymorphic, i.e., it is rewritten to look different in each replicant of the virus.

An example of a virus, which carries this approach to a great length, is the `Idea.6155.A` virus ([Szor98]; see also section 1.2.2). It encrypts a large part of its body using the IDEA cipher ([Lai91]), among other things. Still, the decryption key is present in the body of the virus – so the purpose is not really concealment but variability of the virus body, as well as making it more difficult (and slower) for the anti–virus programs to decrypt the virus, in order to identify and disinfect it.

The decoding algorithm does not always have to be present in the body of the virus. For instance, the `W32/Resurrel.A` virus ([Szor01]) relies on the algorithm used by Windows when loading executable files and fixing their relocation items to decode its body. Still, since both the decoding algorithm and the decoding key are either present together with the encoded message or widely known, this is encoding and not real encryption.

It should be emphasized that the purpose of this approach is *not* to conceal the contents of the virus body from an intelligent attacker (cryptanalyst). Its purpose is only to make the virus look different in every infected file and make it difficult for the scanners that rely on scan strings to detect the virus reliably.

In practice, among most contemporary anti–virus programs, the exclusive reliance on scan strings became obsolete more than a decade ago. Nowadays other, much more powerful methods for virus detection and identification are used. Nevertheless, most people (including the virus writers) still think that we rely on scan strings, which is why encoded viruses are still rather popular.

1.1.2 Data Extortion

Some computer viruses and other malware use cryptographic means to scramble the data of the user and to "hold it hostage" – either trying to extort some kind of payment from the victim or trying to punish it for removing the virus.

One of the first pieces of malware using this approach was the `AIDS Information Trojan` (it was not a virus; see [Gliss90] for a description of it). Its author distributed it as a diskette, which claimed to contain a program that provided information about AIDS, allowed the user to evaluate the risks of getting the disease, and so on. In reality, the program provided none of this. Instead, it simply encrypted the hard disk of the victim that ran it and demanded that a payment be made to its author, in order to "register" the program and to receive the decryption key in exchange. Fortunately, the encryption algorithm was extremely weak. The anti–virus researchers quickly devised utilities, which could decrypt the encrypted disks and restore the data to the users.

Another typical example of a virus that uses this approach is the `One_Half` virus. It is a multi-partite virus (it infects both files and boot sectors). When infecting a machine, the virus slowly begins to encrypt its hard disk on a sector-by-sector basis, starting from the end and moving backwards. (After the virus has finished encrypted the half of it, it displays a message, thus its name.) While the virus is active in the memory of the machine, it decrypts the encrypted sectors on the fly as the user is trying to access them – so, the damage it not immediately apparent. However, the virus keeps the decryption key in its body. If the virus is removed (e.g., by an anti–virus program), the key is lost. Again, the encryption algorithm is weak (XOR with a 16-bit key) and can be broken easily using a known-plaintext attack – but this is not something the average user will be capable of doing and it is difficult to do it in a completely automated way.

Yet another example of such a virus is the `W32/Crypto.A` virus, which uses the `RDA` polymorphic engine (see also section 1.2.2) for its decoder, but also uses Microsoft's Crypto API to encrypt the DLLs that are accessed on the infected machine. While the virus is active in memory, it decrypts the encrypted files on the fly – but if the virus is removed, they will remain encrypted and the system will not work. It should be noted that the encryption used is cryptographically strong.

1.1.3 The Disk Encryption Virus

A variant of the idea described in section 1.1.2 has been implemented in the `KOH` virus ([Bontchev94]). This is a boot sector infector, which encrypts the disk of the machine that it has infected using the strong IDEA cipher ([Lai91]). The purpose of this, however, is not to hold the user's data hostage but to protect its confidentiality. The encryption key is based on a password requested from the user.

Essentially, this is just a disk encryption utility with a viral infection mechanism added to it, in order to help it spread and become more popular among the users. Naturally, since most users would be reluctant to run voluntarily a virus on their machine, in practice this has had just the opposite effect – the author of this paper is not aware of anyone who is intentionally using this "utility".

1.1.4 Authenticated Login to Backdoors

Many of the thousands of known backdoor Trojans (some of them installed by viruses) leave the infected machine wide open to anyone who knows its IP address and the port, on which the backdoor is listening. However, some take the additional care to allow access only to the author of the malware – not to just about anyone.

This is achieved by various forms of authentication – from the use of simple passwords (or passphrases), to relatively complex cryptographic means.

1.1.5 Encrypted Payloads

The `Cheeba` virus was probably the first virus, which used real encryption for its own body ([Gryaznov92]). The virus is memory-resident. It intercepts all file opening operations and calculates two hashes on the last 9 characters of the name of the file being opened, using the following algorithm:

```
unsigned char FileName[9], c;
unsigned short int Hash1, Hash2;
int i;
for (Hash1 = Hash2 = 0x20, i = 8; i >= 0; i--)
{
     c = FileName[i];
     Hash1 += c;
     Hash2 <<= 1;
     Hash2 ^= c;
}
```

If both hashes match the values stored in the virus body (`0x1AE7` and `0x03B7` respectively), the virus uses the last 9 characters of the file name (converted to lowercase) as a cyclically repeated key. The bytes of this key are sequentially subtracted from the corresponding bytes of an encrypted area in the virus, in order to decrypt it. This area contains the payload of the virus. Once it is decrypted, control is passed to it:

```
          MOV    DI, offset Payload        ; Point DI to the start
                                           ; of the encrypted area
Restart:
          MOV    SI,[KeyPtr]    ; KeyPtr points to the beginning
                                ; of the last 9 characters of the
                                ; file name
Decrypt:
          LODSB                 ; Get a byte from the key
          OR     AL,AL          ; End of key?
          JZ     Restart        ; If yes, restart from the
                                ;  beginning of the key
          OR     AL,20h         ; Otherwise convert to lowercase
          SUB    [DI],AL        ; Decrypt a byte from the payload
          INC    DI             ; Point to the next payload byte
          CMP    DI, offset PayloadEnd     ; Finished decrypting?
          JB     Decrypt        ; Not yet - keep going

; Decrypted. Start executing the payload

Payload:
```

Since the key used by the virus to decrypt the payload was not present anywhere in the virus body but was taken from an external source (it was an unknown file name), there was no easy way for the anti–virus researchers what the virus did, besides replicating. At the time, this prompted one anti–virus researcher, Dr. Alan Solomon, to assign “unlimited damage” to the virus – because it was unknown what the virus did and he had decided to err on the safe side and assume the worst possible scenario.

Fortunately, due to the cipher used by the virus being extremely weak, it was eventually possible for the anti–virus researchers to break it and to decrypt the encrypted payload – see section 2.2.3. However, it is easy to envisage a virus a much stronger cipher – one, which is impossible to break by conventional means.

An example of a virus that uses a strong cipher for encrypting part of its code is the `W32/Sobig` virus family ([Ferrie03]). These viruses support “self-updating”. They periodically check a built-in list of Web servers for the presence of a particular file. If found, the file is as-

sumed to contain the URL of another file, which the virus downloads and executes. In practice, these files tend to contain mail proxies or other Trojans – but, in theory, this mechanism could be used to download new, improved versions of the virus. Variants `C` to `F` in this virus family use a variant of DES ([NBS77]) to encrypt the URL. The variant has the first and the last step of the algorithm omitted, which does not reduce the strength of the cipher significantly for the purposes used by the virus (i.e., encryption of short texts).

While at least some part of the replication code of the virus *must* be decrypted (or at least decrypt itself at runtime) in order for the thing to be a virus at all, it is perfectly possible that the payload and/or alternative (e.g., more aggressive) means of replication remain encrypted and concealed in the body of the virus. They will be inaccessible to the anti–virus researchers for analysis, until their decryption is triggered by some external event that supplies the decryption key. Such an event could be the finding of some file name (like in the case of the `Cheeba` virus), the location of a particular sequence of bytes on the hard disk of the infected machine, the receiving of a particular data packet on a communication port, and so on. In other words, the decryption could happen after a specific target is located – or after the virus author supplies the required trigger by broadcasting a specific piece of information in a newsgroup, an IRC channel, or by some other means.

Therefore, it is perfectly possible that, in the future, we see viruses which we are unable to analyze completely – due to a large part of them being hidden behind an encryption level we are unable to break. The paper [Schneier98] contains an excellent discussion of the methods, by which this could be achieved.

However, this is not as ominous as it might sound. Technically, such a virus would be equivalent to those we currently call "programmable viruses" (see also section 1.1.7). Those are viruses, which constantly listen to some communication channel (like monitoring a newsgroup, an IRC channel, or a Web site), download executable code from this channel, and execute it. Many such viruses already exist. It doesn't really matter, from our point of view, whether the inaccessible code is already present in the virus but hidden behind a layer of encryption we cannot break – or whether it is inaccessible because it hasn't been sent yet through the communication channel monitored by the virus.

1.1.6 Encrypted Archives

Since many viruses nowadays spread by mass-mailing themselves as executable attachments, many companies have introduced the reasonable policy of blocking such attachments at their e-mail gateways. Even if they do not do this, they usually run a virus scanner there and scan these attachments for known viruses before allowing them to enter the company's network.

An interesting way around this was invented by the author of the `W32/Bagle` viruses. (The idea itself had been tried before by other viruses, but `W32/Bagle` is the first virus in which its implementation actually works.)

This virus can send itself by the usual means – i.e., as an e-mail attachment with an executable extension, which the user is supposed to click on. However, some variants of it can, in addition, send themselves in ZIP archives – therefore bypassing an e-mail protection policy based on blocking of executable extensions. Since some kind of ZIP archiver (the most popular being probably `WinZIP`) is installed on just about every machine nowadays, and since Windows XP

even has built-in handling of such archives, the users are unlikely to have any problems executing the attachment.

In addition, some variants also use encrypted archives (both ZIP and RAR archives) and communicate the password necessary to decrypt them in the rest of the message. The fact that these viruses have become quite widespread means that no only the users have not learned not to click on executable attachments – but that they are even willing to go through some quite extensive additional steps (starting an archiver, entering a password, then clicking on an executable file in the archive), in order to execute such an attachment. Which only goes to prove that, as noted in [Bontchev01], about 97% of the user population consists of idiots.☺

Although the ciphers used by these two archivers are relatively weak ones (and, in addition, the virus uses very weak passwords from an easy to exhaust keyspace – only 5-digit passwords are used) the encryption level is sufficient to prevent most e-mail scanners from scanning properly the attachment.

Although there are ways for the anti–virus programs to fight back (see section 2.2.4), this approach can be carried much further and can, in practice, devote of its usefulness the concept of scanning the incoming e-mail at the e-mail gateway.

1.1.7 Asymmetric Encryption

Probably the most interesting virus from a cryptographer's point of view is the `W32/Hybris.A@mm` virus ([Nikishin01]). It is a mass-mailing worm which spreads by sending itself as an executable attachment to e-mail messages.

The most interesting feature of this virus is that it is *programmable*. Most of its operation is controlled by independent, replaceable modules, called *plug-ins*. There are currently known plug-ins for it that control the infection of ZIP and RAR archives (by renaming the executable files in them and adding the virus with the original file's name), for updating the set of plug-ins (see below), for displaying a visual payload (an animated spiral), for infection of remote machines compromised by a popular backdoor Trojan (`SubSeven`), for polymorphic encoding, for parasitic infection of DOS and Windows executable files, and for mass-mailing. Several different versions of these plug-ins are known.

The virus can look for new plug-ins (or for newer versions of the plug-ins it already has), expecting them to appear on a particular Web site (`http://pleiku.vietmedia.com/bye/`) . That site is currently not available – but it is not the only place where the virus can look for plug-ins. It also monitors the newsgroup `alt.comp.virus` and collects any plug-ins posted there that it doesn't already have or which are newer from the ones it already has. In addition, the virus itself posts the plug-ins it already has there, so that a single machine infected with a replicant of the virus that has a new plug-in can quickly update all other infected machines in the world that are currently connected to the Internet. Following are two examples of such plug-in containing posts:

```
From:  <Use-Author-Address-Header@[127.1]>
Author-Address: anonymous <AT> anon <DOT> lcs <DOT> mit <DOT> edu
Subject: encr MYLW LWvqPSrKzGPuPSLWnuXGE
Message-ID: <20001210084650.BFKO272.mta8-svc.virgin.net@localhost>
Date: Sun, 10 Dec 2000 08:46:51 +0000
Mail-To-News-Contact: postmaster@nym.alias.net
Organization: mail2news@nym.alias.net
Newsgroups: alt.comp.virus
Lines: 25

SHPCVIUHTXSKPWXLSJQFXJWJMJBLUBOGLXXLWCRERIXDOIRYYCXLLMUCSCSWXESHQM
WYQYNGNZPJLJZCSFMWBXWGOYVKBDRGRLXFXFLEBGNKMCOJWYZZMHQKRIXJOEBKZYSI
YCYKTCPCVXYBTGYXQEMCOXXGTDWKSFRCXJSCXBMXBMTBUBZLWERLNWMXOFUZYBWHPX
TLVDOWXXWYXITYBCTGMCXHUCPETIWCLIBZMKTFPEMIZYMLSCNJUYQZPGLKRZPLSEUJ
XGBWXBBGPKUHXZVYLXSYYFLIVWXZZGWLOLPGUFLJLYOJYBMLXCVLZMYWQBXHRLYCPD
ZCZDBMBXLJQMMWWWTFOMBFRCSLMDZKVZNESGQXULYMMMWYWMVJULXLVEBGSBYKYINL
NDVBVFTBPMWGRKMKMHNKMWVFPHVDYFRZXKMKRDYFXBYCVFLDNMZHZYZEVZTJSLUCSK
OYQXNCMGLYRWVBWJSXBCXLYFLDOLPYYZYJLMPBYLRHLFUGSCPEVKQDBBXZNLOZOGYC
PBPBXFBCUINMMIUWWWRMNLVFSLVIOKBGYFQFXYNWMFOZRBQYBIRWMGYFPHPHWINDTF
PKRBPKMCVETKPYLZZJTYYHRFPCRGQYLKOJBINLTXSMUHMYYBZIZCOXRWMMLXBWUHVY
TGZFNMYDOWRIMFZCMGRXRJYDXBSDBBBGBMLZRBQEBLXFRXVZYWXHNCVIODNHMJXWXH
PMYYRJMXTLVJVIPEULBCUDZYWKSHLBTJNGSXTBBMLKZCBCVZLZUWTYRLRIWMRBSJNB
NZNYRBYMTMTDRBXENLXMYHWJSBZFSJMBXFQZTLYZBGPZZBNCOWTMLLRXTXSWMWPXMY
SHBCWLSGUCUEBKUJSZXEBGBKYIPYLLMDQYUBSIRDUXQLBKSWNZSIUMLMTHTGRKWYOB
QEZDTYYMZBPZYLUWBCQWZDQDQZBLLCMJLCWYSCSWRDSBRWLCMMZDMLPIWZTYZCQJRY
QWUJYJQYMWWBTDQCRIYKRBYZOEVJLJPBMKOFSERKTFVFRMQLQFSIOLYJQCZYLDRBML
YWWZVDTIYWVHODNCVIPDPBNEOYUMNIQKXBMCMBRJYLXHPHZZBFOBYENJMZZDRXZMOI
RDWIBKWGMCNEPLXBPDLJVWNEODMJZCWYWKPYXKXHQCTXWJMMYXYMMILGWBSESEBYBI
SWZXYEVKVGZENMSFXBZIXIMWSXWZYCOGXHBBQJRMOJSCRZVBWGNYMGOGBYWJBMQDWM
OKWCRMQELWQBNIZKQIYBVCLDSEZMLZWKRIWDXBXMTZPZXYYLMMOBOKWKMDRBLYSFUE
ZJRDPLWITGNFMJLJLIRDOHQKUHYFXLQGRGYZLLLXRBMMLJOJVMQLBLPFYIXMWJPKLI
XMLYNMYZNEVCVLBCXCRCYYYBWELMOJMHSHLMQKLFQWMMNHRDRJULNFSMRYPILEVEMI
XFBWMWBIWDNIYXNEVMTIQCPCXBMXVYQYYDUYYJOEXGMCVHOYPJSWSETZUYZHMZZCTL
SWXJPBMWNBZGRIOWTBSZTCNCNLSKNJLMSKYEZCMYYKMGRIZILC
****
```

Example 1

```
From:  <Use-Author-Address-Header@[127.1]>
Author-Address: anonymous <AT> anon <DOT> lcs <DOT> mit <DOT> edu
Subject: text DNDM DmzOziDqLKfOraTCvKbOzmfCnmnqfSTCS
Message-ID: <200012101635343.SM00160@localhost>
Date: Sun, 10 Dec 2000 20:06:09 +0200
Mail-To-News-Contact: postmaster@nym.alias.net
Organization: mail2news@nym.alias.net
Newsgroups: alt.comp.virus
Lines: 28

RZDUIXRCHMGYEVFTIVDSMPEVMRHCSCRYKPGOSTGOLZRVSTJULWHXHWMODPLRMVSQLZ
RTDMRMIRQYQYNXLPMXOYHPJVORQBMVEWFVSZLCNXLVIPEZJXOSOSPYSVOWFTPCMXGC
MBEMKOSNJRLOJNKOESQYFBOUPNFYIBMRHSPNOTPQMSEMMONPJTRUPSETMVNSJXONJW
SWQCJCPYLCDWFONQNXERINIOJODCJTQWFCJVQNRPOYKSFZSCIVDSIRMQPMJPQMPUMV
GURVNUGXJVPYMTFOHQSOFBIOKNHRFUHCKPSOEUEBMPNUHZEZLPFCMRIXSVHYDSIXIQ
OPHOEZHQMURTRQDMIYLUJPKMKCDOJQLTQZGZITNNJMFWLMIWOBEVDWPVOSPVRTMYRP
FNISDZLTMSQNLZRYHZPZSCKTSBHYDMKVSBJPPUNVRCGTOULSSCFWPRFNRYDBFYRSEM
PBQNNXHYPMHYIUJWFWHNNYFROPLSSTIPQCQQPBNQSNMQORLSICMQFOHUEMFNKTRXFZ
HBMOJTQRKZQMNYJOETLOIOHMQTOCENGRDBMBSZGQHBDBQPONIPSMRPEBINKPHYNRRM
JWORJCFMGWGZPXHSMMOROUDCEXJNETGOFWRWRCOTENQWNNKWIPIPGXNRLYKCLWMVFZ
MTKCFOQWIUOYFOESGPGCLXQBDRIUDWDMSWETQOIZKZGWGUNTKBMCSCSMLQMQKSLNRC
EUGBDOFRFRNQQZNQRVFSEUFPLOENJUPQQPSUOZONNUQUGSKMFNKTNBEUFBGWNSRXKW
MPEMQQHXOXOWKZLPOVNPDQFRDSOYPTKSRBJUSXQOIPGNSSMSHOMBHCKQPQHXERLPMU
DUGRFZSVPQDPGCORGNFYHYJVEOENJWKUEBJQQMNYQWDUHPLSKNSQMPORDSFBIXPMDV
PRLYLNISSPPRDUMNPCGXRTDWPPHYHSIPHVFUKYESNWGTKCJOOYSCJTRTDYRBQPIXLS
HYHTQPJRPZQXSTPQSUHNPOLZNZDBOXLVRNEQKUJMKCSOPBLURMRSHTIRMSOCLUETIT
QSIPQMROMBNQNVQZPRJTSZFXIQKMKQQPESDTOYQTKSLQJOQBLSGOJUQTHOKPSMQCMS
SQPMLMSULTSVLMSVSNNYNUOBSTGUHCDRDPFTQXIVLXFXKBHBGQICFBIUMZJSFWSPLB
QCHYMQQTOMRNGULYJZMMQRQSLXLRRRQPGMEVQUGXQNQRJNIPHUDVSPLTFYPCEBDWOS
DMLMJPIPECDVJTJBLCNTHQKUIXDRQUQZKZHTIMDBIPLZLVMSLRDMDNOTQYGOJNKZDC
SOORHPLUEOMSOWKSLBKVSZMOFBMPPSHBHRIWNSNWECRZOTKVOTIYMVPMNTGTMWRWKB
MRFWSWRYLZMZFQFNGCJRFUEMSTLPJSRVHXDQHPOBPRJMOCNCOPOYMNLUGNOUFQLZLZ
MOKWGTLSKQLVJNEUGZGQFZETHSDTNRGYJSNOMTNZNTRXOSEOSWJMLYPVMWNWPXDXMM
PMSPPRNYDQMWLVJOIMKQIRRPFSKCMOSYJUPCNVGVPCJZPBEZPMKOMNOPIQGXDXLOOO
RCPRNBEZRMQCLVHMGVJYGMQZLMHRIOKTSNKYSYKNITRYQYLWFPDRHVOCJQQTORORLQ
QCOWPCLVLSHOFTNORSEUDUIZFOOXEOMUERRMJQECFCJOITRMOZGTOZHRLZLRFXJPGM
HMRQHTGVPNES
****
```

Example 2

The plug-ins are posted encrypted with the XTEA cipher ([Needham97]), using 8 different keys. The first key is generated by computing a cryptographic hash of the plug-in's body, by using the modified Davies-Meyer function ([Schneier94]) with XTEA as the block cipher. The other 7 keys are randomly generated. Then each first 64 bits of the plug-in are encrypted with XTEA using the first key, each second 64 bits are encrypted with XTEA using the second key and so on.

The plug-ins are also digitally signed. A cryptographic hash of the plug-in's body is computed by using the modified Davies-Meyer function with XTEA as the block cipher to produce two different 64-bit hashes which are then combined into a 128-bit hash. This 128-bit hash is then digitally signed, using a 1023-bit RSA ([RSA78]) secret key. The virus contains the public key necessary to verify the signature. This makes it impossible for somebody who does not control the corresponding secret key (i.e., somebody other than the author of the virus) to post "fake"

plug-ins – e.g., a plug-in that would instruct all instances of the virus to remove themselves from the machines they have infected.

It is a bit surprising to the author of this paper that so far we have seen only a single virus that uses such a feature. Given how prevalent the mass-mailing worms currently are, we should expect the virus authors to exploit more thoroughly in the future the idea of "updateablility" through cryptographically protected and authenticated communication channels.

1.1.8 Error-Correcting Codes

Although not strictly related to cryptography, an interesting method was implemented for the first time in the viruses of the `Yankee_Doodle` family. These viruses use self-correcting Hamming code to recover automatically up to 16 bytes of random corruptions in their bodies.

There are multiple reasons behind this approach. *First*, it helps preserve the integrity of the virus by protecting it from random corruptions during its replicating process. It might seem counter-intuitive, but computers *do* make mistakes occasionally – if very rarely. When combined with hundreds of thousands of replications a virus might perform during its lifespan, this significantly increases the probability that the body of the virus will become corrupted. And, indeed, thousands of the known virus variants (especially in the macro virus class) have been seemingly produced by such random corruptions. We cannot know how many other such corruptions there have been – which we have never learned anything about, because they have damaged the ability of the virus to replicate and the resulting program, not being a virus, has never reached us.

Second, it helps preserving the work of the virus author from intentional (albeit incompetent) modifications. Many "wannabe" virus writers do not have the skill, required to write a virus themselves. So, instead they take somebody else's virus and make minor, insignificant changes in it – often as small as patching a text message or replacing a few instructions with a hex editor. Naturally, this tends to annoy the author of the original virus – and self-correcting codes are one way to protect from such trivial patching. Of course, it cannot protect from competent intentional modification by somebody who understands how the self-correcting code works. However, the people who patch trivially other people's viruses usually do not fall in this category.☺

Third, this approach tends to make debugging the virus more difficult. When the person using a debugger instructs the latter to place a breakpoint in the program that is being debugged, the debugger inserts at that place an `INT3` instruction (it occupies a single byte) and then intercepts this software interrupt – in order to be invoked when the debugged program reaches that point. However, if the self-correcting code is executed before the breakpoint is reached, it will "repair" the original code, removing the `INT3` instruction placed there by the debugger and restoring the original contents of the byte that this instruction had occupied. Consequently, the debugger will never receive control at the intended breakpoint.

Since then, several other viruses have used this approach – for instance, the viruses from the `RDA.Fighter` family (see section 1.2.2). However, the approach is not very popular among the virus authors – probably due to its relative complexity and little contribution to the ability of the virus to spread.

1.2 Cryptanalytic Methods Used in Computer Viruses

Computer viruses use very few cryptanalytic methods and do it relatively rarely, so this section will be relatively brief.☺

1.2.1 Key Snatching

The destructive payload of many viruses consists of snatching passwords, encryption keys or other decryption or authentication data and sending this data to the author of the virus. In this section, we shall give just a few examples:

- The `W32/Dumaru.A@mm` virus steals the .PWL files used by Windows to store the cached login information from the Web sites the user has logged to.
- The `W97M/Caligula.A` virus ([VB99]) steals the secret keyring used PGP for Windows.
- The `W97M/Melissa.M@mm` virus (and some others) steals the files `tree.dat` and `smdata.dat`. These files are used by the Windows ftp client CuteFTP (versions 1.x and 2.x respectively) to store the login name and password for the saved ftp accounts. The password is "encrypted" by XOR-ing it with the key `0xC8`, so the information is easily retrievable by the attacker. The popularity of such malware forced the authors of CuteFTP to change the way they were storing this information in versions 4.x and later.
- The `VBS/LoveLetter.BD@mm` virus steals the file `contract.dat`. This file contains the account and PIN information from the on-line banking application used by UBS. This file could then be used in a fresh installation of the UBS software to allow that installation to act as the victim from whom the file was stolen.

1.2.2 Brute-Force Cryptanalysis

The Russian-made viruses from the `RDA.Fighter` family ([Daniloff97]) are very interesting both from the point of view of the anti–virus researcher and from that of the cryptographer. They contain a lot of interesting features. For instance, they are both encoded and encrypted.

The encoding is the outer layer, using variable keys and polymorphically-generated decoders. It consists of up to 16 sub-layers, each of them with a polymorphic decoder, and using various anti–debugging tricks and other armoring techniques to make the decoding difficult for the anti–virus researcher (or the virus scanner). However, the most interesting from the cryptographer's point of view part lies beneath this outer layer.

Beneath it, the virus is additionally encrypted using a proprietary encryption algorithm. The algorithm itself is not cryptographically strong, although it is strong enough to thwart casual cryptanalysis. The most interesting part is the fact that it is real encryption – not encoding. The virus uses a 16-bit key but does not store this key anywhere in its body.

Of course, at runtime the virus must be able to decrypt itself, in order to execute properly. Without having access to the decryption key, the virus achieves this by using a combination of brute force and known-plaintext cryptanalysis.

Essentially, the virus generates random 16-bit keys and uses them to decrypt a few instructions from its encrypted body, then computes a checksum of the decrypted code and compares it with a known value, in order to determine whether the decryption key has been guessed correctly. If it has been, the virus uses it to decrypt the rest of the encrypted part.

It is important to note that the purpose of the virus writer has been again to make the virus difficult to scan for and disinfect – *not* to conceal its contents. When executed in real-time, the virus manages to exhaust completely the relatively small keyspace for its encrypted part relatively quickly – in just a few seconds on a reasonably fast machine. The user is unlikely to notice a few additional seconds of delay as the infected program is started. However, a virus scanner that has to emulate the complete decoding and decryption algorithm of the virus, in order to reach stable, non-variable code that can be identified easily, would need several minutes to scan every infected file.

In addition, the virus encrypts random parts of the infected file and decrypts them before control is transferred to the original program. This tends to make disinfection difficult – because the anti–virus program would have to decrypt these parts when it removes the virus – and this tends to be a difficult and time-consuming process.

The virus also has many other interesting features. For instance, it uses self-correcting Hamming code to restore up to 16 bytes of corruption of its body (see section 1.1.8), it is multipartite (infects both files and boot sectors) and so on.

Another virus that uses a similar technique is the `Idea.6155` virus already mentioned in section 1.1.1. It uses an outer level of encoding with a polymorphic decoder, followed by a middle, encrypted level using one (randomly selected from a set of four) trivial proprietary algorithm, and an inner, final level, encrypted with the IDEA cipher with a 128-bit key ([Szor98]).

The part relevant to this section is the middle level. While the four proprietary ciphers used are extremely trivial from a cryptographer's point of view (they are based on simple XOR, ROR or SUB operations respectively), the key used by them (or even information about the actual algorithm used) is not kept by the virus. Instead, the virus brute-forces this encryption level, using the fact that the keyspace is very small and easy to exhaust completely. The virus uses a keyspace of 16, 8, 32 and 64 possible keys for each one of the four ciphers respectively, so it can exhaust very quickly the whole set of 262,144 possible keys and determine the actual key and cipher combinations, necessary to decrypt the middle level.

2. Cryptographic and Cryptanalytic Methods Used in Anti–Virus Software

In the second part of this paper, we shall examine the various cryptographic and cryptanalytic methods used in the anti–virus software.

2.1 Cryptographic Methods Used in Anti–Virus Software

In this section, we shall examine the various cryptographic methods that are most commonly used in the anti–virus software.

2.1.1 Encoding

As mentioned in section 1.1.1, originally the anti–virus products used scan strings to detect computer viruses. Once the variably encoded viruses mentioned in that section started to appear, there often was only one possible scan string that could be picked for each one of them – a sequence of bytes from the only constant part of the virus, the decoder. Naturally, in such cases all anti–virus products would pick one and the same byte sequence as their scan string for that virus.

This caused an unexpected problem. Since the early virus scanners also scanned the executable files completely for the virus scan strings they used, they would also find the only possible scan string in other anti–virus products that were using it to detect the same virus, therefore causing *false positives*.

The solution was easy and obvious – although it took the anti–virus producers some time before they all started using it. It consisted of encoding the scan strings by using some trivial algorithm and decoding them on-the-fly just before using them. (Again, we would like to emphasize that this is simply encoding, because both the decoding algorithm and "key" are present together with the encoded message – although many anti–virus people use – incorrectly – the term "encryption" for this approach.)

This problem and its solution are mentioned here for historical purposes only. Nowadays the virus scanners rarely rely on scan strings and even when they do, they practically never scan the whole executable file for their presence (they look for them only at the places where they are likely to be present, if the file is indeed infected with the virus which the scan string is used to detect).

Nevertheless, encoding or some other form of obfuscation is still widely used to conceal the data used by the anti–virus programs for the purposes of virus detection, identification and removal. The reason here is not so much to prevent false positives by other anti–virus products, as to conceal from the virus writers what exactly is used to detect their viruses and to make it a bit more difficult for them to produce viruses that would not be detected by the virus scanner.

2.1.2 SSL-Encrypted Communication Channels

Since known-virus scanners can, by definition, detect only known viruses, they must constantly be kept up-to-date. That is, they must be updated regularly, in order to make them capable of detecting the new viruses that constantly appear at the rate of several thousands each month. Most popular anti–virus packages nowadays have the capability to update themselves on-line. Many of them even can do it automatically, at times scheduled by the user.

For the producers of commercial anti–virus products, these updates are a significant source of income. In fact, for some of them they are the *main* source of income – the anti–virus product itself is often distributed free of charge but the customer has to pay for the updates.

Naturally, such companies need to ensure that only paying customers can download the updates they have paid for. This is achieved by using various forms of authentication – many of them cryptographic by nature. In addition, many companies encrypt the whole communication session with the customer, using the encryption provided by SSL (the Secure Socket Layer) and the https protocol.

2.1.3 Sending of Virus Samples

In order to fight viruses more effectively, the anti–virus researchers who trust each other to a sufficient degree often send each other the new viruses that they receive. This is usually done by e-mail.

Since e-mail is an inherently insecure medium (as it travels in plaintext across the hostile Internet and passes an unknown number of computers which are outside the control of both the sender and the recipient of the e-mail message), some steps need to be taken, in order to ensure the confidentiality of the messages that contain viruses. This is necessary both in order to guard

against a malicious attacker (although this is less important nowadays when computer viruses are so prevalent and easily available from various virus exchange sites to anyone who wants some), and to prevent accidents if the e-mail containing the virus is delivered by mistake to a wrong person instead of to its intended recipient.

Encrypting the viruses that we send to each other is the obvious way to solve these problems – which is why we have been doing it from the beginning. Initially, CARO (the Computer Anti–virus Researchers' Organization; an informal group of technical, trustworthy, and highly competent anti–virus researchers from all over the world; a group which the author of this paper belongs to) used encrypted ZIP archives and a 8-character password that was changed regularly. However, since the weakness of the PKZIP cipher was proven in 1994 ([Kocher94]), we have started using PGP instead.

Nowadays using PGP for this purpose is considered state-of-the-art and is widely used not only by CARO but also by many other anti–virus groups and companies who send virus samples to each other.

2.1.4 Encrypted Storage

At the beginning of the computer virus "era", some people advanced the idea that protection against computer viruses could be achieved by keeping the executable code on some kind of encrypted storage – i.e., in encrypted files or on an encrypted disk partition. The idea was that at the time the virus is running, it would not be able to access the protected executable code (because it is encrypted) and that even if a virus manages to "infect" an encrypted file, the decryption process would effectively corrupt it, resulting in its failure to run properly and, therefore, warning the user that something was amiss.

Unfortunately, advances in computer virus technology proved this idea as being a flawed one and nowadays it is almost never used for the purposes of protection against computer viruses. There are several reasons why the idea failed.

First of all, we must observe that no matter how much of a computer's disk is encrypted, that computer must be able to *boot*. Therefore, at least some part of the disk – the master boot record (MBR) – must be unencrypted (and must be loading the decryption software, in order to gain access to the encrypted part of the disk). However, there are computer viruses that infect the MBR. Therefore, disk encryption cannot protect against at least that class of computer viruses.

Second, the virus writers have started creating *memory-resident* viruses. Those are viruses which remain constantly active in the memory of the computer, intercept some file- or disk-related function (e.g., opening a file or reading a disk sector) and, if the object being access is executable and infectable, they infect it. In practice, this means that such viruses would be able to gain access to the files on an encrypted disk partition as they are accessed and, therefore, decrypted. In other words, they will be able to "see through" the encryption. Even if the encryption is implemented at a file (instead of at a disk) level, such viruses would still be able to infect the encrypted files as they are accessed – although the infection is likely to corrupt them.

Third, the virus writers came up with a technology, called *stealth*. Originally, this technology was invented in order to thwart the anti–virus programs. The stealth viruses are memory-resident viruses that intercept the operating system's function for accessing executable objects (e.g., file or sector reading). If the object being accessed is found to be infected by the virus, the stealth virus modifies the information returned by the function it has intercepted, so that the ob-

ject looks uninfected to the caller of that function. Clearly, if such a virus has infected an encrypted file and is active in the memory of the machine accessing this file, the decryption software will not even notice that the file is infected. The file will look perfectly normal to it and will decrypt and execute without problems. Therefore, even file-level encryption cannot guard against such viruses.

2.1.5 Integrity Checking

Computer viruses are essentially an integrity problem – not a confidentiality problem. Therefore, the most powerful means to protect against them is by using tools that ensure the integrity of the protected information. This is much more effective than, say, using access controls or looking for known viruses. Nevertheless, looking for known viruses is the most widespread method for virus protection nowadays – despite the fact that it is the weakest one. (It is trivial to thwart it – the virus writer simply needs to create a new virus, unknown to the anti–virus software of the victim.)

The reason for this is because the users, in general, have a very low level of competence in the area of computer security. Known-virus scanner give them reports which they can easily understand – "File Foo.exe is infected with the virus Bar.A" As opposed to that, integrity-oriented tools produce reports which are too generic and not easily understandable by the users – e.g., "File Foo.exe has been modified". It is not always clear whether the modification has been caused by a virus, or because the file is self-modifying, or because the file has been replaced with a newer version.

Still, integrity-oriented anti–virus programs do exist (they are called "integrity checkers") and they are occasionally used. The idea is to use them to take some kind of "snapshot" of a known clean state of the system and then periodically to use them in order to detect whether the executable objects on that system have changed their contents.

The obvious way to create such a snapshot is to compute some kind of checksum of each executable object on the system and to store the collected checksums in some kind of database, which should be used subsequently for verifying whether these checksums have changed, possibly as the result of a virus infection.

A professional cryptographer would probably jump to the conclusion that the "proper" way to compute these checksums is by using cryptographically strong hashes like MD5 ([Rivest92]) or SHA. Surprisingly, this would be a wrong conclusion.

As demonstrated in [Radai91], it is not necessary to use a cryptographically strong checksum for the purposes of *virus detection*. (Such checksums are necessary only for the purposes of *authentication*.) A simple CRC provides just as much protection against viruses as a cryptographic checksum – and is much faster to compute. The only requirement is that the CRC is computed based on a polynomial, which is not accessible to the virus or to its author. That is, this polynomial must be randomly created for each protected machine and should be stored in a place, inaccessible to the virus.

Although it is relatively easy to "forge" a CRC (i.e., make a modification to a file, so that its CRC does not change) when its computational polynomial is completely known, it is not possible to do so when the computational polynomial is unknown. In those cases, a CRC is just as difficult to forge, from the point of view of the virus, as a cryptographically-strong hash. A stealth virus would be able to bypass the protection (because it will make the infected objects

look uninfected and their checksum will remain the same) – but, then, such a virus would bypass a protection based on cryptographically strong hashes just as easily. Therefore, using a cryptographically strong hash for the purposes of detecting virus infections does not add additional security – it only slows down the computation of the checksums.

It is important to note, however, that although integrity-based virus protections are the strongest kind of defense against computer viruses, many effective attacks against them exist. A full description of these attacks is outside the scope of this paper – but an excellent discussion of them can be found in [Bontchev92].

2.1.6 Public-Key Authentication

As mentioned in section 2.1.2, the known-virus scanners need frequent updating. Naturally, their users need a way to ensure the integrity of the updates. They need to be able to detect both natural corruptions and malicious substitutions by the virus writers. Digital signatures using public-key authentication are ideal for this purpose. In this section, we shall list some examples of how it is used by various anti–virus products:

- FRISK Software's F-PROT (the product made by the company which the author of this paper works for) uses detached PGP signatures to authenticate both its executable files and its virus definitions databases.
- Network Associates' ViruScan uses RSA digital signatures ([RSA78]) to sign its virus definitions files. The scanner contains the code necessary to check the validity of the signatures against a known public key.
- Kaspersky Labs' Kaspersky Anti–Virus uses 512-bit DH/DSS keys to sign its virus definitions files; again, the verification of the digital signatures is performed by code built into the scanner.

2.2 Cryptanalytic Methods Used in Anti–Virus Software

In this sub-section, we shall consider the various cryptanalytic techniques currently in use by the anti–virus software.

2.2.1 The X-Ray Technique

As explained in section 1.1.1, the vast majority of "encrypted" (encoded really) viruses use some simple operation (XOR, AND, etc. with a variable but short key) to make a large part of their body change from one replicant to another. In the early days of anti–virus research, when most scanners were still relying mainly on scan strings, a technique was devised to counter this approach. It was invented independently by several anti–virus researchers (the author of this paper has seen it described for the first time in an unpublished paper of the Russian anti–virus researcher Eugene Kaspersky in 1992) and was known under various names – but perhaps the most common name under which it was known was "the X-Ray technique". A reasonably good description of this technique can be found in [Ciubotariu03], [Ciubotariu04] and [Perriot04].

In essence, the technique consists of applying some constant transformation to the encoded part of the virus body – a transformation, which results in an invariant output, no matter what key has been used to scramble the virus body. In the following description, we use the notation

unit the elementary unit of the virus body on which the scrambling is performed. Usually this is a single byte – although in some cases it can be a double-byte WORD or a quadruple-byte DWORD;

^ the scrambling function. Usually this is an ADD, SUB, or XOR – although NEG, MUL and DIV could also be used;

~ the inverse of the scrambling function – i.e., the function used to descramble the virus body. Some scrambling functions (e.g., XOR) are identical to their inverse;

C_i the scrambled unit at position *i* in the virus body (ciphertext);

D_i the unscrambled unit at position *i* in the virus body (plaintext);

K the key used by the scrambling function, if the key is the same at each position *i*;

K_i the key used by the scrambling function at position *i* in the virus body, if the key is not kept constant during the scrambling/descrambling process;

S the modifier by which the key is changed at each step *i*.

Let us first consider the simpler case of using a constant key throughout the scrambling/descrambling process. We have

$$C_i = D_i \text{ ^ } K$$

for scrambling and

$$D_i = C_i \sim K$$

for descrambling. Now let us consider the transformation

$$C_i \sim C_{i+1} = (D_i \text{ ^ } K) \sim (D_{i+1} \text{ ^ } K)$$

Assuming that the scrambling/descrambling functions have the associative property (which is the case in the functions we have seen used in the viruses that employ such methods of obfuscation), the above can be reduced to

$$(D_i \sim D_{i+1}) \text{ ^ } (\text{^ } K \sim K)$$

and since

$$(\text{^ } K \sim K)$$

is a no-operation, the end result is

$$D_i \sim D_{i+1}$$

which does not depend on K. Therefore, the initial transformation is an invariant one. In other words, if we XOR together (or ADD together, etc., depending on the scrambling function used) the adjacent units of the virus body, we always end up with the same result, no matter what particular key K has been used by that particular instance of the virus. Then simple constant scan strings can be used on the result of our invariant transformation in order to detect and recognize the virus.

Some viruses employ slightly more sophisticated scrambling functions – e.g., functions that vary the scrambling key after each unit of the virus body is scrambled. This variation is almost always trivial – usually by adding or subtracting some constant value to or from the key

used at the previous step. Note that these two methods are essentially the same – since subtracting is the same as adding but with a negative number. In other words, at each step *i* of the scrambling process, the key used is

$$K_0 = K$$

$$K_{i+1} = K_i + S$$

In order to handle such cases, the X-Ray technique carries the invariant transformation one step further:

$$\begin{aligned} C_i \sim C_{i+1} &= (D_i \wedge K_i) \sim (D_{i+1} \wedge K_{i+1}) = \\ &= (D_i \wedge K_i) \sim (D_{i+1} \wedge (K_i + S)) = \\ &= D_i \wedge K_i \sim D_{i+1} \sim K_i \sim S = \\ &= D_i \sim D_{i+1} \sim S \ (\wedge K_i \sim K_i) = \\ &= D_i \sim D_{i+1} \sim S = \\ &= M_i \end{aligned}$$

$$\begin{aligned} M_i \sim M_{i+1} &= (D_i \sim D_{i+1} \sim S) \sim (D_{i+1} \sim D_{i+2} \sim S) = \\ &= D_i \sim D_{i+1} \sim S \sim D_{i+1} \wedge D_{i+2} \wedge S = \\ &= (D_i \sim D_{i+1} \sim D_{i+1} \wedge D_{i+2}) \wedge (\sim S \wedge S) = \\ &= D_i \sim 2 \bullet D_{i+1} \wedge D_{i+2} \end{aligned}$$

The end result is again a sequence of units, the contents of which depends neither on K_0, nor on S and, therefore, conventional constant scan string search techniques can be used on it for the purposes of virus detection and recognition.

Note, however, that the result of applying the invariant transformation to the ciphertext is *not* the plaintext. In other words, the invariant transformation does not result in decoding the virus – it only results in one and the same sequence of units, no matter what particular keys and key modifiers have been used by the particular instance of the virus. The consequence is that this technique is not sufficient for the purposes of virus *disinfection* – because, in order to remove the virus and to restore the original contents of the infected file, the anti–virus program has to restore the original entry point of the file – and the encoded viruses almost always store that in an area covered by the encoding. Therefore, in order to disinfect the programs infected by such viruses, the virus body has to be decoded completely.

Nowadays, the X-Ray technique is mostly obsolete for several reasons. *First* of all, in order to handle complex polymorphic viruses, the contemporary virus scanners use *emulators*. That is, they emulate the instructions of the decoder of the virus body in a virtual environment – until these instructions have decoded the body. Clearly, once one has the decoded virus body, there is no need to apply any invariant transformations before scanning it.

Second, the virus writers have started using scrambling functions, which cannot be handled well by the invariant transformation described in this section. At the same time, they can be handled without problems by simply emulating the decoder of the virus body.

Third, for the purposes of disinfection, one has to decode the virus body anyway.

Fourth, contemporary scanners rely very little on scan strings – they tend to use much more advanced approaches like virus maps and exact virus identification. For instance, the scanner produced by the anti–virus company which the author of this paper works for does not use scan strings at all for the purpose of handling macro viruses – and it uses scan strings only as a

trigger for its more sophisticated virus identification algorithms when handling other kinds of viruses. For such scanners, it is of little importance whether sufficiently long constant sequences of bytes are present in the virus body or not – therefore, they do not need to resort to the X-Ray technique.

However, the technique can still be of considerable usefulness when dealing with some special classes of viruses ([Perriot04]) – the so-called *entry-point obfuscators* (viruses, which transfer control to their body in unconventional ways that cannot be easily traced from the entry points of the infected file) and the viruses that use various anti-emulation techniques to detect that they are being emulated and/or slow down the emulation process. Provided, of course, that these viruses use the simple encoding algorithms for which the X-Ray technique is practical.

2.2.2 Known-Plaintext Cryptanalysis

The programs Microsoft Word and Microsoft Excel from the office automation package Microsoft Office 95 support encryption of the documents they can create. Unfortunately, they also support rather powerful macro languages and are vulnerable to macro viruses. Therefore, the anti–virus products have to scan the documents that these programs create, find any macro viruses present and, if necessary, remove them. However, what to do with the documents that are encrypted?

These office automation packages store their documents in OLE2 files. The format of these files is incredibly complex – essentially, they are whole filesystems-in-a-file – with its file allocation table (FAT), clusters, "directories" (called "storages") and "files" (called "streams"). Word 95 stores both the text of the document and the macros in a stream named `WordDocument`. This stream is almost entirely covered by the encryption. Excel 95 stores the spreadsheet data in a stream named `Book` – and it is almost entirely covered by the encryption. It stores the macro-related information in a whole set of streams and storages – and they are *not* covered by the encryption. Therefore, it is possible to scan encrypted Excel 95 documents for macro viruses without any problems.

However, the `Book` stream contains, among other things, information that there *are* macros in the file. This information is covered by the encryption, which makes it impossible to *disinfect* encrypted Excel 95 documents – removing the macros without removing the information that they are present would corrupt the document.

Therefore, the encryption is an impediment to proper handling (scanning and removal) of macro viruses in Office 95 documents. Fortunately for the anti–virus producers, this encryption is a very weak one. In Word 95, it is a trivial variant of a Vigenère cipher with the slight "improvement" that bytes which contain binary zeroes are not encrypted (because there are lots of them in a normal Word 95 document and they would reveal the key). By necessity, the bytes which have the same contents as the corresponding byte of the key are also not encrypted, because encrypting them (by XOR-ing them with the byte of the key) would result in zeroes, undistinguishable from the unencrypted zeroes. The key used is 16 bytes long and is derived from the password supplied by the user in a rather simple way. More exactly, the encryption algorithm is

```
void CryptBlock (unsigned char *pBlock,
                 unsigned long ulLength,
                 unsigned long ulOffset,
                 unsigned char *key)
{
     register n;
     register unsigned char c;

     for (n = (int)(ulOffset % 16); ulLength--; pBlock++)
     {
          if (*pBlock)
          {
               c = key [n];
               if (*pBlock != c)
                    *pBlock ^= c;
          }
          n++;
          n &= 0xF;
     }
}
```

In other words, the non-zero bytes are encrypted by XOR-ing them with one of the 16 bytes of the key. The encryption used by Excel 95 is only slightly more complicated. There a combination of XOR and ROL (rotate bitwise left) is used:

```
void CryptBlock (unsigned char *pBlock,
                 unsigned long ulOffset,
                 unsigned int ulLength,
                 unsigned char *key)
{
     do
     {
          *pBlock++ = crotl8 ((unsigned char)
                              (key [ulOffset++ & 0x0F] ^
                              *pBlock), 3);
     }
     while (--ulLength);
}
```

Vigenère ciphers are particularly vulnerable to known-plaintext attacks, if the attacker has a piece of known plaintext of length equal to or greater than the length of the key used. Since in this particular case the key length is 16, the problem boils down to finding a piece of known plaintext that is at least 16 characters long. Fortunately, this is relatively easy.

Every Word 95 document contains the string “Default Paragraph Font” somewhere in the `WordDocument` stream. This known plaintext is more than sufficient to recover the 16-byte key. A slight complication comes from the fact that the different language versions of Word 95 store this phrase in their own language (instead of in English). However, we have collected documents created by all the language versions known to us (Czech, German, Greek, Spanish,

Finnish, French, Hungarian, Italian, Dutch, Norwegian, Polish, Russian, Slovenian and English) and in all of them this phrase is more than 16 characters long. The language, which has the shortest phrase is, surprisingly, French. There the phrase is "Police par défaut" – but with its 17 characters (including the spaces) it is still more than sufficient for recovering the 16-byte encryption key.

In Excel 95, there is no such "universally present" phrase. However, even there it is possible to find known plaintext almost always. The `Book` stream consists of variable-length records. Each record begins with a 2-byte record identifier, followed by a 2-byte record length, followed by the contents of the record of the specified length. The encryption covers only the contents of the records – i.e., it skips the record identifiers and the record lengths, so each kind of record can be identified very easily even in encrypted documents. One of these records contains the name of the author of the document – as defined in Excel's Tools/Options/General/User name menu. While in reality most names tend to be rather short (or even empty, if the user has not bothered to enter such information), the record containing this information is rather long, has a fixed size, and is padded to the right with spaces. Therefore, we can almost always find at least 16 spaces at the end of this record and recover the encryption key. In addition, the name of the registered user usually can be found unencrypted in other parts of the OLE2 file containing the Excel document – namely, in the `SummaryInformation` stream.

Since the encryption of Word 95 and Excel 95 is so trivial to break by automated means, many anti–virus products (including the one made by the company which the author of this paper works for) have implemented by capability to do on-the-fly cryptanalysis of the documents produced by these Microsoft Office applications, scan them for viruses and, if necessary, decrypt them.

Unfortunately, starting with Office 97, Microsoft has been making this task progressively more complicated. The cipher used by these versions of Office is RC4, using a 40-bit key taken from the MD5 hash ([Rivest92]) of the password entered by the user. While, from the point of view of a professional cryptographer, such a cipher is still ridiculously weak (it can be broken in a matter of days, using a simple brute-force attack for exhausting the 40-bit keyspace and assuming a speed of about a million RC4 trial encryptions per second), it is strong enough to make breaking it on-the-fly prohibitively expensive for the anti–virus programs. In a sense, the user is put in a lose-lose situation where he gets the worst of both worlds – a cipher, which is too weak for the purposes of confidentiality protection, yet one which is too strong for the purposes of virus detection.☹

One good thing is that in these versions of Office, the macro structures are *not* covered by the encryption – both in Word and in Excel they are stored outside the stream that contains the user data and which is covered by the encryption. This allows the anti–virus programs to at least *scan* encrypted Office documents for macro viruses. However, the information that macros are present in the document *is* stored in places covered by the encryption. As a result, encrypted documents cannot be *disinfected* without corrupting them. Fortunately, the corruption is relatively minor and basically results in a non-critical error message when the document is opened; the data in the document is still intact and accessible to the user.

2.2.3 Frequency Cryptanalysis

As explained in section 1.1.5, the `Cheeba` virus contains a payload, which is encrypted using a trivial variant of the Vigenère cipher. They encryption key is not present in the virus. In-

stead, it is taken from the environment (it is a file name) and is authenticated by using two trivial (non-cryptographic) hashes.

Naturally, the anti–virus researchers were faced with the problem of determining the possibly destructive effects of the virus, hidden behind the encryption. In a rather classic paper ([Gryaznov92]), the Russian anti–virus researcher Dmitry Gryaznov describes how he managed to break the encryption, using a combination of brute-force and frequency cryptanalysis attacks.

Initially, he tried a pure brute-force attack. However, due to the large size of the keyspace (9 characters from the set {a-z0-9.!#$%&'()^_-}), he quickly realized that this was a dead end. His next step was to use some observations on the properties of the algorithms used to compute the two hashes, in order to reduce drastically the size of the keyspace to about $2.56 \bullet 10^{10}$. It was possible to exhaust this keyspace in reasonable time. However, due to the fact that the two hashes were not cryptographic, they matched way too many possible keys in this keyspace.

Then the researcher tried a dictionary attack, scanning the hard disks of many machines for existing file names that matched the two checksums used by the virus. Unfortunately, none were found.

At this point, the researcher decided to resort to frequency cryptanalysis. He computed the frequencies of the bytes in the ciphertext and compared that to a frequency table computed from the unencrypted part of the virus, making the reasonable assumption that the encrypted part, being written by the same author, probably had a similar programming style and similar frequency of bytes. Since the sizes of both the ciphertext and the area used to compute the frequency table were relatively small, the frequency match was not perfect and did not result in immediate decryption.

However, it resulted in decrypting correctly *some* parts of the payload. It also resulted in a tremendous reduction of the size of the keyspace – to only 2,150,400 possible keys, which could be exhausted very fast by a brute-force attack and which yielded only a single key that matched both hashes. The "magic words" were "`users.bbs`".☺ This was a file used to contain the user names and login passwords information of a popular bulletin board software and the payload of the virus consisted of inserting a privileged entry (a backdoor account) in this file.

2.2.4 Scanning Encrypted Archives

As mentioned in section 1.1.6, the virus writers have started creating mass-mailing viruses that send themselves as e-mail attachments. These attachments contain an archive, which archive contains the executable body of the virus and is password-protected (i.e., is encrypted). The password, necessary for the decryption of the archive is communicated to the user in various ways – for instance, in a text message in the body of the e-mail or by various other means. Naturally, this presents a serious problem to the designers of virus scanners oriented towards detection of viruses in e-mail messages. Basically, the problem boils down to how can the anti–virus product detect the virus behind the level of encryption. Various methods have been deployed by the anti–virus producers.

Since the first viruses of this kind (those of the `W32/Bagle` family) communicated the password for the archive in the text body of the e-mail message they sent, the natural initial reaction of the anti–virus producers was to attempt to parse this text body and extract the password from there. For instance, one could try to look for standard phrases like "the password is", or simply try all words and quote-delimited strings in the message body as possible password candi-

dates. One example of an e-mail scanner using this approach is the one produced by Kaspersky Labs. There are, however, several problems associated with it.

First of all, natural language is notoriously difficult to parse. While any particular virus can use only a fixed number of phrases by which it tries to communicate the password to the user, the number of these phrases can be very large and they can vary very widely from one virus to another. It would be very ineffective if the anti–virus producer has to re-design the message parser of their scanner that locates the password every time a new password-using virus appears.

Second, the virus writers have started communicating the password to the user in non-textual means. For instance, the later `W32/Bagle` variants construct a picture (in BMP, GIF or JPG format) containing the password and attach this picture to the e-mail message that contains the encrypted archive with the virus. Although, in theory, optical character recognition methods could be used to locate the password in these pictures, this is extremely difficult – especially if the viruses start employing methods for making OCR methods ineffective (e.g., by introducing random noise in the pictures). Basically, with one simple step, the virus writers have made the life of the anti–virus producers more difficult by an order of magnitude.

However, using a picture is just one of the possible non-textual means a virus could use to communicate the password to the user. Although we have not seen alternative methods used by the existing viruses yet, such methods are definitely possible. For instance, the virus could use text-to-speech algorithms to *pronounce* the password, save the result in a WAV file, and attach this file to the e-mail message containing the encrypted archive. Alternatively, the virus could send the password in a follow-up message. Since the virus scanners scan e-mail messages on a message-by-message basis, they will have no easy way for making the connection between a message containing an encrypted archive and another, seemingly unrelated message to the same recipient, containing the password for it. Clearly, different and more effective methods have to be used by the anti–virus producers.

One possibility is to use brute-force cryptanalysis on the encrypted archives that are found in e-mail messages. For instance, the `W32/Bagle` virus always uses a 5-digit password for the encrypted archives it sends itself in. On a Pentium III machine, a well-implemented brute-force cracker for PKZIP archives can exhaust this complete password space in just about 55 milliseconds. However, the anti–virus producers cannot afford to rely that future viruses will use similarly weak passwords. It is very trivial for the virus writers to enlarge the password space to include any alphanumeric strings of length up to 8 characters or more – thus making brute-force cryptanalysis of the encrypted archives prohibitively expensive.

Another possibility is to use the known-plaintext cryptanalytic technique outlined in [Kocher94] and break the encryption level regardless of the password length used by the virus. This technique requires that at least 12 bytes of the encrypted file are known in advance – but this is not a problem, because once the anti–virus researchers receive at least one sample of the virus (which they need anyway, in order to implement detection of it in their known-virus scanners), the *whole* file will be known.

Unfortunately, there are several problems with this approach, too. For instance, it is comparatively slow. On an average Pentium III machine, it takes about a minute and a half to crack and encrypted PKZIP archive. This is prohibitively expensive time-wise for a scanner designed to scan the incoming messages at the e-mail gateway – because, in a large company, it often has to scan thousands of messages per hour. In a virus outbreak, the slowdown caused by the need to

cryptanalyse each one of the thousands of encrypted archives sent by the virus from all over the world will cause huge delays and will further exacerbate the e-mail denial-of-service attack already caused by the virus. (Basically, again as in the case described in section 2.2.2, the user is in a lose-lose situation. The encryption is too weak to protect the user's data from a determined attacker – yet it is also too strong to allow the user to be protected from viruses that employ it.☹)

In addition, Kocher's attack is applicable only to PKZIP archives. The later variants of the `W32/Bagle` virus already use encrypted RAR archives too. No equivalent attack is known for those – and there are also a variety of other popular archivers (e.g., ARJ, LHarc, etc.) that can easily be used by computer viruses in the future. Clearly, methods that are more efficient must be used to combat this problem.

Various companies, including one, which the author of this paper is working for, have chosen a heuristic approach to the problem. We do not attempt to decrypt the encrypted archives, in order to scan them. Instead, we try to determine heuristically whether an encrypted archive is likely to contain a virus and we flag such archives as suspicious. An example of a heuristic that can be used for such purpose is

```
IF  (the archive is encrypted) AND
    (the archive contains a file with an executable extension) AND
    (this file occupies at least 90% of the archive)
THEN the archive is suspicious
```

In general, we recommend our customers to implement a policy of blocking at the company e-mail gateway all e-mails that contain encrypted archives as attachments – the same way they we are recommending that they block all e-mails that contain executable files as attachments.

Unfortunately, the author of this paper has the feeling that we, the anti–virus people, are fighting a losing war here. There is much room for future development and it favors the virus writers. There are already many viruses which harvest e-mail addresses from various sources (address books of the infected machine, HTML files found there, etc.) and send themselves to these addresses. It is a relatively trivial step for them to contact the PGP keyservers and to determine whether a public PGP key has been published, with a UserID matching the e-mail address found by the virus. If such a PGP key is found, the virus could encrypt itself with it (using the widely available GnuPG library), and send itself as a PGP-encrypted attachment to the found e-mail address. There is absolutely nothing an e-mail scanner can do to scan such attachments – and most people would object strongly to a policy of blocking all encrypted e-mail.

If computer viruses start using such approaches widely, they will make e-mail scanners mostly useless. The only hope for protection will then lie in the on-access scanners on the workstations – scanners, which scan files for viruses just before these files are executed – and a virus has to decrypt at least its replication part at execution time.

Unfortunately, this approach is wrought with a completely different kind of problems – problems, which have caused the e-mail scanners to rise in popularity in the first place. Consider, for instance, a large company with tens of thousands of PCs, attacked by a mass-mailing virus that cannot be stopped at the e-mail gateway, because of the encryption it uses to conceal its attachments. Even if just a virus is stopped by the on-access scanners at the workstations and causes no direct damage, the mere fact that tens of thousands of untrained users will see the alert of the scanner and hundreds, if not thousands of them will panic and call the tech support de-

partment will be damaging enough. Not to mention that panicked users are likely to delete files indiscriminately, that some users are bound to have the on-access scanners on their workstations not installed, not functioning, not up-to-date and so on will exacerbate the damage and will create a rather high probability that the virus spreads within the organization.

3. Acknowledgements

The author would like to thank to his colleagues from CARO (the Computer Anti–virus Researchers' Organization) for the many ideas, suggestions and useful bits of information which have helped him considerably in the writing of this paper.

4. References

[Ciubotariu03] Mircea Ciubotariu, "Virus Cryptoanalysis", Virus Bulletin, November 2003, pp. 6-7.

[Ciubotariu04] Mircea Ciubotariu, "Advanced Virus Cryptoanalysis", Virus Bulletin, September 2004, pp. 6-9.

[Bontchev92] Vesselin Bontchev, "Possible Virus Attacks Against Integrity Programs and How To Prevent Them", Proc. 2nd Int. Virus Bull. Conf., September 1992, pp. 131–141.

[Bontchev94] Vesselin Bontchev, "Are 'Good' Computer Viruses Still a Bad Idea?", Proc. EICAR'94 Conf., pp. 25–47.

[Bontchev01] Vesselin Bontchev, "Anatomy of a Virus Epidemic", Proc. 11th Virus Bull. Conf., Prague, 2001, pp. 389-406.

[Daniloff97] Igor Daniloff, "Fighting Talk", Virus Bulletin, December 1997, pp. 10-12.

[Ferrie03] Peter Ferrie, "Sobig, Sobigger, Sobiggest", Virus Bulletin, October 2003, pp. 5-10.

[Gliss90] Hans Gliss, Ralf Herweg, "A Report On the Panama Diskette", Comp. Fraud & Security Bulletin, February 1990, pp. 6-10.

[Gryaznov92] Dmitry Gryaznov, "Analyzing the Cheeba Virus", Proc. Annual EICAR Conf., Munich, Germany, 7-9 December 1992, pp. 124-136.

[Kaspersky98] Eugene Kaspersky, "DarkParanoid – Who Me?", Virus Bulletin, January 1998, pp. 8-9.

[Kocher94] Eli Biham, Paul C. Kocher, "A Known Plaintext Attack on the PKZIP Stream Cipher", CS 842, December 1994, Proc. of the Leuven Workshop Fast Software Encryption 2, LNCS 1008.

[Lai91] X. Lai, J. L. Massey, "A Proposal for a New Block Encryption Standard", Proc. of Advances in Cryptology, Eurocrypt'90 (1991), I. Damgard, Ed., vol. 473 of Lecture Notes in Comp. Sci., Springer Verlag, pp. 389-404.

[NBS77] National Bureau of Standards, "Data Encryption Standards", FIPS Publication 46, U.S. Dept. of Commerce, 1977.

[Needham97] Roger M. Needham and David J. Wheeler. Tea extensions. Technical report, Computer Laboratory, University of Cambridge, October 1997.

[Nikishin01] Andy Nikishin, "Harnessing Hybris", Virus Bulletin, January 2001, pp. 6-7.

[Perriot04] Frédéric Periot, Peter Ferrie, "Principles and Practice of X-Raying", Proc. 14th Int. Virus Bull. Conf., September 2004, pp. 51-66.

[Radai91] Yisrael Radai, "Checksumming Techniques for Anti–Viral Purposes", Proc. 1st Int. Virus Bull. Conf., September 1991, pp. 39-68.

[Rivest92] R. Rivest, "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RSA78] R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126.

[Schneier94] Bruce Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 1994, ISBN 0-471-59756-2.

[Schneier98] James Riordan, Bruce Schneier, "Environmental Key Generation Towards Clueless Agents", Mobile Agents and Security, Springer 1998, LNCS State-of-the-Art Survey, pp. 15-24.

[Szor01] Peter Szor, "Tricky Relocations", Virus Bulletin, April 2001, p. 8.

[Szor98] Peter Szor, "Bad Idea", Virus Bulletin, April 1998, pp. 18-19.

[VB99] "Et tu, Caligula!", Virus Bulletin, February 1999, p. 4.