Cryptovirology: Extortion-Based Security Threats and Countermeasures*

Adam Young
Dept. of Computer Science,
Columbia University.

Moti Yung
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598.

Abstract

Traditionally, cryptography and its applications are defensive in nature, and provide privacy, authentication, and security to users. In this paper we present the idea of *Cryptovirology* which employs a twist on cryptography, showing that it can also be used offensively. By being offensive we mean that it can be used to mount extortion based attacks that cause loss of access to information, loss of confidentiality, and information leakage, tasks which cryptography typically prevents. In this paper we analyze potential threats and attacks that rogue use of cryptography can cause when combined with rogue software (viruses, Trojan horses), and demonstrate them experimentally by presenting an implementation of a *cryptovirus* that we have tested (we took careful precautions in the process to insure that the virus remained contained). Public-key cryptography is essential to the attacks that we demonstrate (which we call "cryptovirological attacks"). We also suggest countermeasures and mechanisms to cope with and prevent such attacks. These attacks have implications on how the use of cryptographic tools should be managed and audited in general purpose computing environments, and imply that access to cryptographic tools should be well controlled. The experimental virus demonstrates how cryptographic packages can be condensed into a small space, which may have independent applications (e.g., cryptographic module design in small mobile devices).

1 Introduction

Every major technological development carries with it a certain degree of power. This power is often beneficial to society, but more often than not it can also be severely misused. A perfect example of such a technology is atomic fission. Cryptography is a blessing to information processing and communications (as atomic fission is to energy production), because it allows people to store information securely and to conduct private communications over large distances. It is therefore natural to ask, "What are the potential harmful uses of Cryptography?" We believe that it is better to investigate this aspect rather than to wait for such attacks to occur. In this paper we attempt a first step in this direction by presenting a set of cryptography-exploiting computer security attacks and potential countermeasures.

The set of attacks that we present involve the unique use of strong (public key and symmetric) cryptographic techniques in conjunction with computer virus and Trojan horse technology. They demonstrate how cryptography (namely, difference in computational capability) can allow an adversarial virus writer to gain explicit access control over the data that his or her virus has access to (assuming the infected machines have only polynomial-time computational power), whereas from an information theoretic point of view (assuming all parties are all-powerful) this is impossible. This idea is then extended to allow a distributed *virus* to gain itself explicit access control over the information on infected machines, provided it is not detected early enough and vigorously destroyed. This shows that viruses can be used as tools for extortion, potential criminal activity, and as munitions in the context of information warfare, rather than their traditional reputation of being merely a source for disturbance and annoyance. In general, we define cryptovirology to be the study of the applications of cryptography to computer viruses. We note that cryptography has been used to help prevent viral attacks (i.e., by providing strong integrity checks) and to try to hide a virus's structure, yet formal cryptographic paradigms have never before been used successfully as a weapon in viral attacks.

In describing the first set of attacks, a new virus model is proposed. The model is motivated by biological organisms that are capable of modifying the host to depend on the organisms themselves. Such a virus

*Supported in part by NSF grant CCR-93-16209 and CISE Institutional Infrastructure grant CDA-90-24735

0-8186-7417-2/96 $5.00 © 1996 IEEE

forces a symbiotic relationship between itself and its host. Alternatively, this dependency may also be derived from an effect that the virus has on the host, such that only the author of the virus can reverse the effect. As we shall point out, this later situation is a mere approximation to the former.

Preventive measures are described in response to the attacks. They are a step in the right direction to help prevent and recover from such attacks. In fact, it is shown that the public availability of cryptographic tools without proper access control, can put the data on a computer system at serious risk.

1.1 Organization

Background information on viruses is given in section 2; the notion of a computer virus is defined and a set of general rules that pertain to computer viruses is given. We develop cryptovirology from the perspective of survivability. Section 3 introduces a definition of what we believe to be a computer virus with the property of high survivability (i.e., one that forces the host to retain the virus), and its inherent characteristics. Examples of attempts to make the high survivability virus are also discussed.

Section 4 describes a set of cryptovirological attacks which attempt to approximate the high survivability virus (i.e. not being survivable, but having an effect which is survivable unless the virus writer interferes). Rather than having the absolute survivability property, the attacks create absolute dependency on the virus writer. Section 5 presents a virus that attempts to manage its own keys in a distributed fashion based on secret sharing techniques, so there is a dependency of the local user on the (distributed) virus itself. In section 6 we suggest measures that can be taken to reduce the threat posed by cryptovirological attacks and to minimize abuse of cryptographic facilities. The Appendix contains a description of our experimental cryptovirus as well as the virus performance on a Macintosh computer (but no code is described!)

2 Background

The notion of a Trojan horse was first discovered by D. Edwards and was described in the Anderson report [And72]. It is a program that resides in another program that does something that the user is unaware of. A Trojan horse may, for instance, reside in a compiler and transmit source code to the author of the Trojan. One of the early Trojans was a binary code segment that was inserted into Multics binary code that was distributed to all sites [KS74], thus demonstrating the feasibility of and difficulty of finding a Trojan horse. To guard against such attacks it is necessary to confine programs into small domains with only the rights that are needed for their functioning and to guard their integrity (either in binary or source code modes). Lampson showed that as long as a borrowed program does not have to retain information, confinement can be achieved by restricting access rights to the program. However, most of the time covert channels are available for leaking information [Lam73].

Computer viruses are similar to Trojan horses since they remain hidden from the user. Some viruses are benign and merely consume CPU cycles, while others overtly delete and overwrite data. We will not present a rigorous definition of a computer virus here. Instead, we will adopt the following definition put forth by Fred Cohen. A computer virus is defined as a program that can infect other programs by modifying them to include a, possibly evolved, copy of itself [Coh89].

The fields of virus and antivirus technology are broad in scope and are slowly changing over time. Perhaps the latest development in the field of virus technology is the advent of Polymorphism. A virus that is polymorphic creates offspring with object code that is different from that of the parent. The original computer viruses were monomorphic in nature. That is, their object code remained essentially the same across viral generations. Polymorphic technology was developed in response to viral scanners which use databases of search strings to identify known viral strains.

Viral polymorphism, among other things, involves the encipherment of the main body of the virus to make it difficult to detect. Typically, the XOR operation is used with a randomly chosen value to accomplish this. The resulting ciphertext is in no way secure from simple cryptanalysis, but makes the task of identifying the main body of the virus using fixed-string scanners more difficult. We do not include weak methods of "virus self-encryption" as part of cryptovirology.

There are several rules that all viruses seem to obey. (1) By virtue of being programs they all consume CPU time and occupy space. Also, (2) since viruses need to gain control of the program counter in order to execute, they must (directly or indirectly) modify code in the host system in order to do so. The last and perhaps most interesting rule of viruses is (3) their inherent vulnerability to user scrutiny. Viruses can always be frozen and analyzed by the user. They can be backed up (or a backed up copy can be found) and later scrutinized in detail using a low level debugger. In what follows we show that this vulnerability can be effectively bypassed if strong cryptographic techniques are employed and if the virus acts fast enough, i.e. before detection.

3 High Survivability Virus

We are interested in making the host dependent on the virus. Thus, we design cryptovirology from the point of view of survivability. That is, a virus can survive in the host if it makes the host depend in a critical way on the very presence of the virus itself. If we cannot achieve this, we may approximate it by writing a virus such that its effect on the host is only reversible by the virus writer (so the dependence is approximated by making the host depend on the author rather than the virus). In section 3.1 we define a highly survivable computer virus (or h-s virus), and conclude with a few examples of past attempts to implement it. We also discuss the possibility of its existence. We then describe an approximation to high survivability, namely, having an effect that is survivable and thereby leaving the host at the mercy of the virus writer. We analyze the possibility of the survivable effect (or damage) with respect to Information Theory and Computational Complexity. Note again that we are interested in an effect that gives an advantage to the virus writer rather than having an effect where no one can help (like viruses that simply erase files for annoyance).

3.1 Survivable Virus

Virus writers try very hard to make their viruses difficult to detect, since they know that users will try to remove them once they are found. Polymorphic, Stealth, Mutation, and Tunnelling technologies are a virus's best means for evading detection in systems that run antiviral software. The high survivability property of a computer virus that we propose is one that forces a non-beneficial symbiotic relationship between a virus and its host so that the survival of the virus is essential to the survival of the host[1].

Survivability is an issue common to viruses, Trojan horses, worms and other forms of "malware." We will use "virus" in the discussion below to represent all of these. The following is our definition of a computer virus with the "high survivability property".

Definition 1 *A computer virus has the "high survivability" property if it can maintain control over a critical host resource R_c such that it grants access to R_c solely when it is needed, and such that if the virus is modified or removed, R_c is rendered permanently inaccessible.*

Note that this definition implies that the virus has modified the machine's state to such a point that if you rid the machine of the virus you lose access to the resource. This is a definition which is optimal from the virus writer's point of view. We stress that it is not necessarily achievable– but if a virus can achieve such control, and if the resource it holds "hostage" is crucial, then the virus will survive in the host.

Four notable rogue programs have appeared in the wild that seem to reflect the intention of remaining resident after detection. These programs are the One-Half virus, the LZR virus, the AIDS Information Trojan, and the KOH virus.

The One-Half virus operates by encrypting the hard drive starting from the last cylinder and slowly moving forward over time. The One-Half virus uses a symmetric cipher, and stores the secret key within itself. To rid the host of the effect of the virus, the key can be obtained from the virus code, and the damage undone.

The LZR virus is even closer to a h-s computer virus. LZR takes control of reads and writes to the hard disk using a relatively unknown system call [DB95]. LZR writes error correction information to the disk, even though error correction is not usually performed by the operating system. As information is written to the disk, the data is followed by the error correction data of the viruses' choosing. If the virus is removed, the viral routine will not be called, and the files will be rendered incomprehensible to user programs. The damage caused by LZR can be undone by copying all of the damaged files to floppy disks and then disinfecting the virus with an appropriate antiviral program. This disinfection works because the error correction routine is not invoked when writes are made to floppy disks. Even if this error correction mechanism worked with floppy disks, it would be possible to write an antiviral program that would repair all the data over a period of time.

Though not a virus, the AIDS Information Trojan nonetheless exhibits traits similar to that of a h-s computer virus. It provides information on the users risk of contracting AIDS, and at the same time encrypts the users hard drive after 90 reboots. The user is then informed that a license fee must be paid in return for the decryption key [Sla94]. This Trojan is one of the first extortion attempts made using rogue programs. Unfortunately, we do not know the exact cipher used by the AIDS Information Trojan. It can be considered a step in the direction of a h-s virus.

The KOH virus is a virus that is used to encrypt the data on a host system. The motivation for the virus is to allow encryption to be performed in the background, so that user intervention is not required. This virus incorporates the use of the IDEA cryptosystem and is sold commercially. It will be shown later that such a virus containing a symmetric cryptosystem cannot be

[1] This is analogous to H. R. Giger's fictional "facehugger" that appeared in the movie "Alien" [BS79]; the unfamiliar reader who is interested in the gory details may see the movie.

used to mount extortion-based attacks.

3.2 Exploiting Intractability

An approximation to a h-s virus is a virus having a survivable (or lasting) effect. In other words, after detection the computer cannot rid itself of the effect of the virus unless the virus writer helps. This opens avenues for serious attacks by viruses. We next explore survivability and survivable effects using two frameworks: Information Theory and Computational Complexity.

Claim 1 *From an information theoretic perspective everyone has the same ability to recover from the damage caused by a virus (in particular, if the virus writer can recover from the damage caused by a given virus, then so can everyone else, and if no one else can recover from the damage caused by a given virus then the writer cannot recover as well).*

Proof Let S denote a machine (or system) and let V denote a virus capable of infecting S. We assume that V's program is available (it is well understood) to the victim as well as to the writer. This is so, since from an information theoretic point of view, once the virus program is available its effect can be fully understood (as was demonstrated in [ER88]).

Assume that S is initially uninfected. V is either obtrusive or non-obtrusive. Consider the case in which V is non-obtrusive, meaning that it does not interfere with the current state or the stored states of S. In this case, given S, we can always get the current (later) state with or without the virus being present in this computation. Since the virus is non-obtrusive it follows that everyone can just remove the virus. Now consider the case where V is obtrusive and modifies the states of S. For example, V may erase or XOR the current state and all previous states stored on S. If a backup of a previous program and state cannot be provided, then it is impossible for anyone to undo the damage caused by V to the data on S. Or more formally, the distribution of the correct state given the damaged state is (equally) available to everyone (the writer as well as the victim). Note that if there is a backup of the state prior to infection, we can restart the computation and we are effectively in the non-obtrusive case again.

It therefore follows that from an information theoretic perspective, either everyone can recover from the damage caused by V, or no one can recover from the damage caused by V (not even the writer). □

Claim 1 was proven based on Information Theory – that is, if the victim is infinitely powerful he can still either recover, partially recover, or not recover at all. The virus either does not change the state, changes S in a way that it is easily recoverable, changes S such that one can use a backup to recover, or changes S such that there is a probability distribution of initial states– and in the worst case the initial state is not determinable.

In all these cases, from Information Theoretic perspective, the virus writer cannot affect the outcome in this matter. Now we consider the case in which the computational resources of the victim are polynomially bounded.

Claim 2 *From the perspective of Computational Complexity, there are cases where a virus can cause damage such that the victim cannot recover, but the virus writer can.*

Proof We assume the case that no backups exist (if there are backups then clearly the victim can recover). The proof of this is by exploitation of the assumed strength of public-key cryptography (which was used before to break symmetries based on infinite computational power of parties assumed by Information Theory[2]).

This is easy to achieve by supplying the virus with a public key. The virus can encrypt data D on the host machine S with this key. Evidently, from the definition of a Public Key Cryptosystem and from the fact that only the virus writer knows the private key corresponding to the public key of the virus, the claim holds. □

Note that we have shown a way to bypass the last rule of computer viruses that we presented earlier. The virus contains trapdoor information such that this information does not reveal itself when the virus is scrutinized. Does this solve the problem of making a virus with the high survivability property? We shall present evidence that indicates that it does not.

Let F be a h-s virus, D be a critical data file in host system S that contains several (software and hardware) components, and U be the user of S such that $F, D \in S$. Clearly, the user is external to the machine, thus, $U \notin S$. Let R be a relation on the set of elements consisting of the user U, the virus F, and the data D, where $(x, y) \in R$ iff x can encrypt and decrypt y. Our goal is to have (F, D) be contained in R and at the same time have (U, D) not in R. We shall see from the following three cases that either (F, D) is not in R, or that (U, D) is in R by transitivity of the encryption/decryption capability relation.

[2] In [SRA79] (see also [Sch96] pp. 93) it was shown that dealing fair hands from a deck of cards by two parties is impossible from an information theoretic point of view and possible based on computational complexity via cryptographic assumptions.

1. F contains a secret key and tries to monopolize critical resource D. From the way F is constructed, it is clear that (F, D) is in R. Since U can read and control the data on S, and since F resides on S, the (U, F) is in R. Since $\{(U, F), (F, D)\}$ is in R, (U, D) is in R by transitivity. It follows that U can always decrypt D' to get D. Since F cannot prevent U from having access to D, F is not a h-s computer virus.

2. F contains a public key and no private key. F can then encrypt D to get D', but cannot decrypt D', therefore (F, D) is not in R. It follows that F cannot monopolize critical resource D, and is therefore not a h-s computer virus.

3. F contains a public key and its corresponding private key. It follows that F can transform D into D' and vice-versa, thus (F, D) is in R. However, since (U, F) is in R, U has access to the public and private keys in F. It is therefore the case that (U, D) is in R by transitivity. Since F cannot prevent U from having access to D, F cannot be a h-s computer virus.

Intuitively, it makes sense that implementing the h-s virus is a difficult task. A h-s virus must not only be immune to the scrutiny of all users, but must also be able to control access to D. We leave the possibility of the existence of h-s computer viruses as an open problem.

Note, however, that in case 2 the virus writer accomplishes something unusual. If F manages to encrypt D to get D' and if U does not have a backup of D, then only the virus writer will be able to transform D' back into D. This act breaks the symmetry between what the user has access to and what the virus writer has access to as claimed possible in Claim 2. A first order approximation to the h-s computer virus is therefore possible since the virus can do damage that the victim cannot repair, but that is possible to fix. This means that an adversarial virus writer can gain explicit access control over user data by having a survivable effect.

Adleman has shown that detecting viruses is an intractable problem, and that it seems unlikely that protection systems predicated on virus detection will be successful [Adl90]. His approach towards computer viruses was from the perspective of Computability, whereas our approach is based on Computational Complexity. We have shown that even if a virus is detected in a given system, it may be a computationally intractable problem to reverse its effect on the host system (assuming public-key cryptography is strong).

4 Cryptovirological Attacks

In this section a series of cryptovirological attacks that use the above observations are presented where the possessor of the private key of the virus is the author.

4.1 Survivable and Reversible Cryptographic Attack

We define a cryptographic attack to be a denial of service attack using a public key. The attack is survivable unless the virus writer reverses it. A cryptographic attack can be performed by a cryptovirus or a cryptotrojan, which are defined by the following.

Definition 2 *A cryptovirus (cryptotrojan) is a computer virus (Trojan horse) that uses a public key generated by the author to encrypt data D that resides on the host system, in such a way that D can only be recovered by the author of the virus (assuming no fresh backup exists).*

The setting for the denial of service attack is as follows. Immediately following encryption, the cryptovirus notifies the user and demands that the user contact the virus writer. Once contacted, the virus writer demands a ransom in return for the private key. Once the private key is obtained, the user is able to decrypt D' to get D (assuming no backups). A drawback from the perspective of the virus writer is that he cannot free one victim without potentially freeing all the victims, because the freed victim could publish the private key. This drawback can be solved if the virus contained multiple public keys. The virus could randomly (or otherwise) choose a key for a given attack, thereby allowing the virus writer to free some victims without freeing all the rest. This gives the virus writer more control over who he can selectively free. However carrying many keys is expensive. Also, note that this is only a partial solution, since users may cooperate with each other. Another drawback is the fact that encrypting a file directly with a public key is slow.

To solve the above problems we will employ hybrid cryptosystems, in which the session key is used to encrypt the critical data. If the cryptovirus generates a large random session key and encrypts this key with the public key for each machine that it attacks, then with very high probability, each victim will need a different key to get their information back. In this case the adversary never discloses his private key to the users. Instead, he demands that they give him the ciphertext of the session key (and whatever else is necessary for decryption). For a suitable ransom, he will decrypt the session key for them. Also the encryption will be fast.

The hybrid cryptographic attack is a reversible denial of services attack. It is reversible since by communicating with the victims, the virus writer is able to return the data that is denied. This contrasts with the traditional notion of a denial of service attack in which, for example, data is permanently deleted.

The following is a more detailed description of the hybrid cryptographic attack. A cryptovirus is equipped with a strong random number generator and a strong seeding procedure and mounts an attack by generating a random session key K_s and a random initialization vector IV. Let the public key in the virus and private key of the writer be denoted by K_f and K_w respectively, where f denotes the virus and w denotes the writer. The plaintext message $m = \{IV, K_s\}$ is formed by the virus, and is encrypted with K_f to get the ciphertext $m' = \{IV, K_s\}K_f$. The virus then encrypts the critical system data using the K_s, IV, and a symmetric algorithm. A suitable mode of operation of the symmetric cipher is output feedback mode (OFB) [Den82, Sch96]. After encryption the original file is overwritten.

The next part of the attack is aimed at getting the user to contact the virus writer. The virus prints a message to the screen containing m' and the phone number of the virus writer. If the victim has a back-up of the data file, then the victim need not contact the virus writer. The victim may then try to disinfect the system. If there is no back up, then the victim must contact the virus writer to retrieve the original data. Upon making the phone call, the virus writer asks for m' in addition to a suitable ransom. Once the ransom is paid, the virus writer decrypts m' using K_w. The session key and IV are then given to the victim. Since the victim never finds out K_w, he is unable to assist other victims of the cryptovirus with high probability.

Remark 1: Sending the information to the author does not necessarily reveal the author. The information may be asked to be posted publicly while being encrypted using the public key of the attacker. A public bulletin board can be used for such purposes. Unlike physical resources, information resources do not need to be shipped to the attacker, again due to the power of cryptography.

Remark 2: Stealing attack: We can use a cryptovirus to securely steal information from a remote location and use the viral spread as the communication medium. Rather than announcing its presence after data D is encrypted, a cryptovirus can simply append D' to itself (perhaps without affecting the local copy). The virus then replicates as usual but kills any ancestors or siblings that it encounters that do not already contain D'. If the author is lucky he will encounter an offspring with D' and decrypt it. The virus is a secure communication medium since no one else, except the writer, has the ability to decrypt D'.

We end this section with a description of a general purpose cryptotrojan that is capable of compromising system security while minimizing the authors risk of getting caught. Packet sniffing and keystroke monitoring Trojan horses are a well known method of stealing passwords. Such Trojan horses typically store the pilfered passwords in a hidden file. The drawback to these Trojan horses is that the passwords that are hidden are at risk of being found by system administrators. In addition, the attacker must either make the passwords publicly accessible or must later log in to download the passwords. A cryptotrojan using the author's public key solves both of these problems and provides safe storage for the stolen information.

4.2 Information Extortion Attack

The information extortion attack is an attack in which the virus writer is able to force the victim to exchange information in return for the session key and IV, and in addition provides a mechanism for verifying the *authenticity* of the data being extorted. This attack can only be carried out successfully if the virus succeeds in encrypting critical information that cannot be replaced by the victim. Systems that manipulate up-to-the minute valuable information are highly susceptible to such an attack. This attack uses the hybrid cryptographic attack, with a few modifications. This attack can extort resources, but can also be used as a tool for espionage and information warfare.

The information extortion attack permits a virus writer W to demand a file of arbitrary size by including a checksum of that file in the message block m. In this attack, W demands the desired file in addition to the ciphertext of the message block m. The following is the data structure for the message block:

$$\begin{aligned} m &= \{ChkSm, IV, K_s\} \\ m' &= \{ChkSm, IV, K_s\}K_f \\ K_f &= \textit{public key of } W \textit{ in virus } F \\ K_w &= \textit{private key of } W \\ ChkSm &= \textit{checksum of file desired by } W \\ IV &= \textit{random initialization vector} \\ K_s &= \textit{random session key} \end{aligned}$$

The only value above that is not strictly random is $ChkSm$, which is a function of the session key, IV, and the file that the virus writer desires. The attack works as follows. F is written by W and is programmed to look for critical data D and desired data H. Upon migrating to the correct host system S, the virus mounts a hybrid cryptographic attack in the following way. It first generates IV and K_s, randomly using its built in

random number generator. It then uses a symmetric cipher to encrypt D, and overwrites the original file. F then looks for data H. If it is found, a cryptographic checksum of it is performed to get $ChkSm$. This is accomplished by using H, IV, K_s, and a symmetric cipher (say, in CFB mode [Den82, Sch96]). The message m is then formed by F and encrypted with K_f to get m'. After m is overwritten in RAM, the virus displays m' to the user and instructs the user to contact the virus writer and upload m' along with H. This completes the information extortion attack by the virus.

Provided the user follows the prescribed protocol, he sends W the ciphertext m' and data H. The virus writer then decrypts m' with K_w and extracts IV, K_s, and $ChkSm$. W then performs a checksum on H using IV and K_s and compares the result to $ChkSm$. If the checksum matches, W assumes H is authentic and sends $\{K_s, IV\}$ to the user. Otherwise, W assumes that the user altered H and does not send him $\{K_s, IV\}$.

Suppose that the user wants to cheat W. If the user doesn't want K_s then he can send W an H of his choosing and its corresponding m'. He can do this by simply choosing H and performing the attack himself. It follows that the attack will only be successful if D is a critical resource that is not backed up. Now suppose that the user wants $\{K_s, IV\}$ but doesn't want to give the virus writer H. He could do this by modifying H and either sending him m' or an altered m'. If he sends m' along with an altered H, the virus writer will detect this when he computes the checksum of H. Now consider the case where the user sends a modified H and modified m'. In this case, since the user does not know K_w he does not know how to alter m' correctly and with very high probability will be caught when W computes the checksum of H. Even if the checksum succeeds, chances are he will receive the wrong K_s. Any forgery by the user must be done without knowledge of the session key, IV, or checksum value. The assumption being made here is that the session key and IV are not captured by the user during the short period of time in which they are in RAM.

The information extortion attack could translate directly into the loss of U.S. dollars if electronic money is implemented. In fact, the potential for attacks on anonymous e-money has been recognized in the cryptographic literature [vSN92, BGK95, SPC95, JY96]; we materialize an attack via a cryptovirus. A specialized cryptovirus could be designed to search for e-money notes and encrypt them. In this way, the virus writer can effectively hold all the money ransom until half of it is given to him. Even if the e-money was previously encrypted by the user, it is of no use to the user if it gets encrypted by a cryptovirus. Electronic money must therefore be treated with great care, since it is subject to ransom on any machine that is subject to viral attack.

Appendix A contains a description of our experimental cryptovirus. The virus is a Macintosh virus which performs the information extortion attack on a specific date. The virus is under 7k bytes in length and requires a total of 12 seconds to complete its attack. It contains code for RSA, the Tiny Encryption Algorithm [WN94], and truerand [MB95].

The cryptographic engine of the virus represents a beneficial application of our work: the demonstration of a space efficient cryptographic module (applicable to small devices like mobile units).

5 The Secret Sharing Virus

In this section we show how to implement a virus that is a very close approximation to a h-s virus. Whereas in the above attacks the virus author managed the keys and owned the private key, here the virus itself will manage its private key. This sounds paradoxical, since a virus holding a public key and managing its private key can be analyzed and could lose its power. However, this is accomplished by changing our notion of a system S to be a network of computers, and to regard the host as being the entire network. We use the distributed environment to hide the key in the virus copies themselves.

Let us describe this in some detail. We have shown how Public Key Cryptography can be used in a virus to encrypt information such that the user cannot retrieve it. In order to be able to decrypt D' to get D, the private key must also be stored somewhere, since otherwise D' cannot be decrypted. We cannot store the entire private key at one node in the network, since this would give the user of that node the entire private key. By considering an entire network as a host we effectively divide and conquer the power of the user, since we now have many different users who do not have access to each others data. The secret sharing virus takes advantage of this property by sharing its private key among m nodes, where $m > 1$. The virus therefore exploits the access controls that users place on themselves to keep its private key secret.

The idea is that a virus will spread itself around the network, and may act autonomously or wait for outside control to act as an agent of the writer. Note that the local users my wipe out parts of the virus (assuming they have back-ups), but then the total network may be damaged (since we need the entire virus pieces to recover). It may therefore be useful for the virus to immediately notify the local machines that if they rid

themselves of the virus they may cause global problems and ask them to first consult with the network's administrator. Alternatively, if we are afraid of making the virus's attack irreversible, we can reserve the option for the writer to know the keys that the virus generates. This can be accomplished by having the virus copies make the private keys they generate available to the the author by encrypting them using the authors public key and publishing these encryptions. This makes sure that at worst we activate the attacks above that require the writer's involvement.

We will now explore some of the subtleties involved in implementing a secret sharing (or key splitting) virus. Consider an ancestry of secret sharing viruses that make use of one public/private key pair. A virus that resides on node i cannot decrypt D' to get D without exposing the entire private key to the user of node i. It follows that the user of node i would know the private key from then on. In order to securely deny access to future data, the viruses must therefore be able to generate other public/private key pairs. This is not an easy problem, since the viruses cannot start with the private key and then split it up. This problem is similar to a traditional problem in secret sharing. How can a group of people share a secret such that no one knows the secret until it is reconstructed? This implies that an arbitrator cannot be used to divide up the secret. Ingemarsson and Simmons demonstrated a protocol that accomplishes this [IS91]. Frankel demonstrated a way to do this using a scheme similar to ElGamal [Fra90]. Both of these protocols are for applications in which a secret is shared statically, i.e. it is split up only once. Our secret sharing virus splits up a secret with each generation. Since the virus replicates exponentially, a dynamic tree-like structure is used to generate keys used in denial of service attacks. Our virus is based on the ElGamal cryptosystem [ElG85].

We will now explain how the secret sharing virus operates. Consider m viruses on m nodes that want to generate a public/private key pair ($m > 1$). All the viruses share the same g and p, such that g is a generator modulo p, where p is a large prime. Each virus generates a random value, denoted by x_i, such that x_i is less than p. The virus on node i then calculates y_i, using $y_i = g^{x_i} \ mod \ p$. The viruses then publish their y_i's anonymously over a public channel (and perhaps they also publish the encryption under the writer's key of their private choices – if we choose this option). Upon reading the values from the channel, each virus computes $y = \prod_{i=1}^{m} y_i \ (mod \ p)$. Every virus therefore has the public key y, g, and p. The private key x can only be found by obtaining all of the x_i's and computing, $x = \sum_{i=1}^{m} x_i \ (mod \ p-1)$. Furthermore, since each node i can only access x_i, none of the users can calculate x without collaborating with the $(m-1)$ other users. For availability purposes a key may be kept at a number of places (and by the writer if we so choose). The network of m viruses therefore has the ability to generate an arbitrary number of public/private key pairs, such that the private keys are shared secretly. To free all m nodes each virus must publish its x_i so that the private key can be calculated by each virus (this is what is shown in [Fra90]).

A dynamic distributed virus: Consider the case in which a set of viruses are spreading on a network. Let N be the set of nodes on the net, with $|N| = n$. Assume that some users trust others and some don't. Also assume that N can be partitioned into users who maintain backups and users who don't. Initially, m nodes are infected with the secret sharing virus. Each virus is programmed to infect exactly two other nodes. In a given round, the viruses generate and publish their y_i's anonymously through a public channel (e.g., a bulletin board). Each virus then computes y and produces children. Each of the children is told whether it is an L child or an R child, where L and R denote left and right, respectively. The children of node p are sent to nodes randomly chosen from the set $N - \{p\}$. A cryptographic attack is then performed on each of the original m nodes using the public key. The users are then notified of the presence of the viruses.

At this point each of the victims will need the other $m-1$ x_i's in order to get the information back. Since the viruses disclosed their y_i's anonymously, each victim has no idea which of the $(n-1)$ other nodes contain the x_i's that are needed. Perhaps some victims will try to locate each other. Since some users are untrustworthy, they might try to give bogus x_i's. If two victims are competitors or enemies they may not help each other at all. Users with adequate backups will be able to restore their own information, and may choose not to publish their x_i. Either way, chances are that not everyone will get their information back every time.

Once resident, the L and R viral children generate new x_i's and calculate the corresponding new y_i's. The L children then take it upon themselves to decide whether or not to repair the damage caused to the original m nodes. The R children were not given the original x_i's by their parents. Each of the L children flips a coin. This toss is then used to determine whether or not to publish the old x_i. In any case, all of the new y_i's are published. If all m of the L children decide to post the old x_i's, then the m original victims will be able to get their information back. Otherwise, either a subset of them or none of them will be given the x_i's necessary to repair the damage. Both the L and R children then calculate their own public keys, and another round commences.

If after a sufficient number of generations, one of the users manages to catch a virus before it infects, it will be at that users discretion to help the previous victims. If m of the victims of a given viral generation get involved in the protocol, then they will have to collaborate to reverse the damage. The secret sharing virus is made possible because each subsequent viral generation decides whether or not to free the previous generation, and because the private key only appears at a given node when the key is needed. The virus permits four possible outcomes for each node of the network:

1. The node is never inhabited by the virus
2. The node gets infected, but recovers after a certain length of time
3. The node gets infected and never recovers
4. The node contains the virus but suffers no damage since backups exist

Remark: A beneficial use of our ideas is organizing the storage of keys so that they are shared but users do not get direct access to them. By exploiting the tree structure it is possible to propagate the keys and split them into two pieces each time. The internal nodes forget their keys but remember the pointers. This assures a careful storage of sensitive keys that is not easily traceable. It can be done with ElGamal public keys in the method that we have described above.

6 Suggested Countermeasures

There are several measures that can be taken to significantly reduce the risk of being infected by a cryptovirus, and there are also measures that can insure a quick recovery in the event of an attack. Fortunately, many of the attacks described in this paper can be avoided using existing antiviral mechanisms, since cryptoviruses propagate in the same way as traditional viruses. The first step in this direction is implementing mechanisms to detect viruses prior to or immediately following system infiltration. One of the pioneering works in the area was "An Intrusion-Detection Model", by Dorothy Denning [Den86]. The paper by White, Chess, and Kuo entitled "Coping with Computer Viruses and Related Problems" is another good source regarding the virus threat [WCK89].

Access control to cryptographic tools: More specifically, we suggest auditing access to cryptographic tools - This is perhaps the major issue that needs to be learned. This will help system administrators identify suspicious cryptographic usage.

Note that if strong cryptographic ciphers and random number generators are made available to user processes, then they will also be made available to cryptoviruses. Such viruses would be smaller than our cryptovirus since they would not contain as much code, and they would also run faster since such tools are usually optimized for speed. Incorporating strong cryptographic tools into the operating system services layer may seem like it would increase system security, but in fact, it may significantly lower the security of the system if the system is vulnerable to infection. Furthermore, with such tools readily available, virus writers would not even have to understand cryptography to create cryptoviruses. Note that this rule should not apply only to export control (as it is now) but also to protection of an installation by its own administration.

On-line proactive anti-viral measures: A general suggestion for an on-line network-wide method for fighting viruses is in "How To Withstand Mobile Virus Attacks" by Ostrovsky and Yung [OY91]. This paper describes a mechanism whereby a network of processors can cope with network viruses. It is shown how local computations (at each processor), and global computations can be made robust using a constant factor resilience and a polynomial factor redundancy in computation. This defense mechanism is of particular relevance to cryptovirological attacks because it allows computations to proceed in the presence of cryptoviruses, and also allows automatic recovery of user data. While the original suggestion is theoretic in nature, a more practical adaptation of this mechanism was suggested in Spirakis et. al. and is called "securenet" [SKG94]. This approach can also be approximated by conducting frequent backups and by employing highly responsive and active anti-viral tools that execute perpetually.

7 Conclusion

We have shown how Cryptography can be used to implement viruses that are able to mount extortion-based attacks on their hosts. Public-key cryptography is essential in enabling the writer to get an advantage over the victim. We also presented an experimental cryptovirus that accomplishes this (it demonstrates cryptographic implementations requiring small space). A model based on a distributed network was then formulated and an algorithm was provided for how to write a virus that is able to gain discretionary access control over its host. We also suggested a set of measures that can be taken to minimize the possibility of and the risks posed by the cryptovirological attacks.

Acknowledgements:
The first author would like to thank Mark Reed and Stephen Clausing for their influence on his career, and Matt Blaze for a security course where some of the material used in this work was introduced to him. We would also like to acknowledge the help of Matt Hastings and discussions with Dave Chess, Greg Sorkin and Steve White. Helpful remarks by Tom Van Vleck and Paul Karger improved the presentation of this work.

References

[Adl90] L. Adleman. An Abstract Theory of Computer Viruses. In *Advances in Cryptology—CRYPTO '88*, volume 403, 1990. Springer-Verlag.

[AHU74] A. Aho, J. Hopcroft, J. Ullman. *The Design and Analysis of Computer Algorithms*, page 280, 1974. Addison-Wesley.

[And72] J. Anderson. Computer Security Technology Planning. study, ESD-TR-73-51, volumes I and II, USAF Electronic Systems Div., Beford Mass. Oct. 1972.

[BS79] D. Bannon, R. Shusett. Alien, (a movie), directed by Ridley Scott, Twentieth Century Fox Film Corp., 1979.

[BGK95] E. Brickell, P. Gemmel, D. Kravitz. Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change. Proc. 6-th ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 457–466, 1995.

[Coh89] F. Cohen. Computational Aspects of Computer Viruses. In *Computers & Security*, volume 8, n. 4, pages 325–344, 1989.

[DB95] H. Delger, N. Brown. comp.virus newsgroup and *private communication*, March, 1995.

[Den82] D. Denning. *Cryptography and Data Security*, page 147, 1982. Addison-Wesley.

[Den86] D. Denning. An Intrusion-Detection Model. In *IEEE Symposium on Security and Privacy*, pages 118–131, Apr. 1986.

[ER88] M. Eichin and J. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In *IEEE Symposium on Security and Privacy*, pages 326–343, 1989.

[ElG85] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology—CRYPTO '84*, pages 10–18, 1985. Springer-Verlag.

[Fra90] Y. Frankel. A practical protocol for large group oriented Networks. In Advances in Cryptology—EUROCRYPT '89, pages 56–61, 1990. Springer-Verlag.

[Has] M. B. Hastings, *private communication.*

[Has90] J. Håstad. On Using RSA with Low Exponent in Public-Key Network. In Advances in Cryptology—CRYPTO '85, pages 294–305, 1990. Springer-Verlag.

[IS91] I. Ingemarsson, G. Simmons. A protocol to Set Up Shared Schemes Without the Assistance of a Mutually Trusted Party. In *Advances in Cryptology—EUROCRYPT '90*, pages 266–282, 1991. Springer-Verlag.

[JY96] M. Jacobsson, M. Yung. Revokable and Versatile Electronic Money. In *ACM, 3-d Comp. and Comm. Security.* 1996.

[KS74] P. A. Karger and R. R. Schell. Multics Security Evaluation: Vulnerability Analysis. ESD-TR-74-193, volume II June 1974 HQ Electronic Systems Division, Hanscom AFB, MA 01731.

[Knu69] D. Knuth. *Seminumerical Algorithms*, volume 2, section 4.5, 2nd edition, 1981. Addison-Wesley.

[Lam73] B. Lampson. A Note on the Confinement Problem. In *Communications of the ACM*, volume 16(10), pages 613–615, Oct. 1973.

[LMS] J. Lacy, D. Mitchell, W. Schell. CryptoLib: Cryptography in Software. AT&T Bell Laboratories, section 2.2.1.

[MB95] D. Mitchell, M. Blaze. truerand.c, AT&T Laboratories, 1995.

[OY91] R. Ostrovsky, M. Yung. How To Withstand Mobile Virus Attacks. In *10th Annual ACM Symposium on Principles of Distributed Systems*, Aug. 1991.

[RSA78] R. Rivest, A. Shamir, L. Adleman. A method for obtaining Digital Signatures and Public-Key Cryptosystems. In *Communications of the ACM*, volume 21, n. 2, pages 120–126, 1978.

[Sch96] B. Schneier, *Applied Cryptography* 2nd edition, John Wiley & Sons, New York.

[Sla94] R. Slade. *Robert Slade's Guide to Computer Viruses*, pages 8, 53–54, 454, 1994. Springer-Verlag.

[SRA79] A. Shamir, R. Rivest, and L. Adleman, *Mental Poker.* MIT Tech. Report.

[vSN92] S. von Solms, D. Naccache, On Blind Signatures and Perfect Crimes. In *Computers and Security*, volume 11, pages 581–583, 1992.

[SKG94] P. Spirakis, S. Katsikas, D. Gritzalis, F. Allegre, D. Androutsopoulos, J. Darzentas, C. Gigante, D. Karagiannis, H. Putkonen, T. Spyrou. SECURENET: A network-oriented intelligent intrusion prevention and detection system. In *Proceedings of the IFIP SEC94*, Curacao, June 1994.

[SPC95] M. Stadler, J-M. Piveteau, J. Camenisch. Fair Blind Signatures. *Advances in Cryptology— EUROCRYPT '95*, pages 209–219, 1995. Springer-Verlag.

[WCK89] S. White, D. Chess, C. Kuo. Coping with Computer Viruses and Related Problems. International Business Machines Corporation, 1989.

[WN94] D. Wheeler, R. Needham. Tiny Encryption Algorithm (TEA). In *Fast Software Encryption: second internation workshop*, volume 1008 of Lecture Notes in computer science, Dec. 1994. Springer.

A Implementation and Performance of the Cryptovirus

Precomputation: Two precomputations were performed to create the cryptovirus. The only other form of virus that we are aware of that requires precomputation are those that decompress themselves during runtime. Such viruses save on disk space and are therefore less noticeable. The first precompuatation we performed was the generation of a public/private key pair. The key modulus used was 512 bits-long. Once a pair was found, the values for p and q were then tested further to ensure primality. We used the GNU MP function mpz_probab_prime_p() with 300 repetitions for testing both p and q. This MP function is an implementation of the primality test in Knuth [Knu69], where it is indicated that 25 repetitions are sufficient.

Space efficient arithmetic implementation: The second precomputation was calculating the reciprocal of the RSA modulus. The virus is equipped with an RSA exponent, modulus, and reciprocal of the modulus. Though the inclusion of the reciprocal is not necessary, it significantly reduced the size of the MP library code in the virus. Since the virus already knows the reciprocal of the modulus, it need not invoke a division routine during RSA encryption. The GNU source files mpn_div.c and mpn_lshift.c were not needed in the cryptovirus as a result of this optimization. The compiled object files of both of these source files comprised 3,808 bytes (which were saved).

The algorithm: A special algorithm is used to perform the modulo operation during RSA encryption. This method is a modification of the division algorithm based on repeated subtraction. We want to compute $a^b \ mod \ c$, without using any divisions. We are given inv and inv_exp, which form the reciprocal of the modulus n as defined in Aho, Hopcroft, and Ullman [AHU74]. The significant bits are represented by inv, and inv_exp is the exponent used to indicate the location of the decimal point. Furthermore, we are given that b is 3 (since we only employ one key we are not exposed to weaknesses of small-exponents when the same message is encrypted by various ciphers [Has90].) We first square a, to get a number x. We then multiply x by inv, and adjust the result using inv_exp. Call this new number t. If x minus t is greater than or equal to c, then we subtract c from x. We then check if x is greater than or equal to c, and subtract c from x if it is, etc. This process is continued until x is less than c. The result of the modulo operation is x [Has]. We then multiply x by a and the same modulo operation is repeated again. The resulting value is $a^3 \ mod \ c$.

Spread prevention: The cryptovirus was designed to only propagate on MC68030 Macintoshes with ROM version 120. This includes the Mac SE/30, Mac //cx, etc. The virus could very easily be modified to infect any Macintosh with the MC68020 processor and higher. The virus was tested with system 7.1 and was developed in two separate parts. It consists of an attacking routine which contains the modified MP library, and a viral routine. Upon completion, the attacking routine was appended to the end of the virus, forming a cryptovirus. The viral routine was written completely in Motorola 68000 assembly language. The virus is programmed to attack on August 13, 1995. It also has a time limit after which it no longer infects systems. Our cryptovirus does not bypass heuristic antiviral programs or activity monitors. Its sole purpose is to demonstrate the information extortion attack.

Virus operation: The virus is similar to TSR viruses found on IBM PC compatible computers. It exists in one of three states at any given time: in a program, in the system file, or in a patch to an operating system routine. When an infected program is run, the virus gets control before the host program and checks to see if the system file is already infected. If it is not infected, the system file gets infected. Control is then sent to the host by the virus. Once the system is rebooted, the virus in the system copies itself into RAM and modifies the trap dispatch table so that the table invokes the resident copy of the virus whenever a program is run. The next time a program is run, the virus that resides in the patch will see if the program is already infected with the cryptovirus. If it is not infected, the virus will attempt to infect it.

If the machine is rebooted on August 13, 1995, the virus in the system file will perform a hybrid cryptographic attack. The virus first generates 384 random bits using its built in random generator. This generator is based on truerand.c by D. P. Mitchell, and M. Blaze from AT&T [LMS]. These bits form the initialization vector and two TEA keys. The virus then computes a cryptographic checksum of the file entitled 'payroll' in the System Folder, provided the file is present. The checksum is performed using TEA in CFB mode [Den82]. The MP library is then invoked and the plaintext is encrypted using RSA. The virus then attempts to encrypt a file entitled 'e-money' in the System Folder using triple TEA in ECB mode. The

triple encryption is performed using the first TEA key, followed by the second TEA key, followed by the first again. This operation overwrites the original file. The virus then overwrites the RSA plaintext key in RAM, and creates a file entitled 'VIRUS DAMAGE' in the system folder. This file contains the RSA ciphertext and information on how to contact the virus author.

truerand physical RNG: truerand produces physically random numbers. It operates by setting an alarm and then incrementing a counter register rapidly in the CPU until the alarm signal occurs. The contents of the register is then XORed with the contents of an output buffer byte. After each byte of the output buffer is filled, the buffer is further processed by doing a right circular shift of each character by 2 bits. This moves the most random bits into the most significant positions. This process is repeated until a truly random number is achieved. The values are physically random since they are derived from the difference in pulses generated by the CPU crystal and the timer interrupt crystal. It thus seems infeasible for a victim to try to calculate the random values derived by the virus after an attack.

Performance:

First note that overall about 10 minutes worth of CPU time was spent on the above precomputations. The following is a summary of the performance of the cryptovirus. The reason for giving an approximate running time is that the value varies from program to program. Factors such as pending disk I/O contribute to the variation.

Table 1
Running Time

system boot (no attack)	$<$	16.7 msec
infect a program	$\approx$	1 sec
infect file 'System'	$\approx$	4 sec
perform RSA encryption	$=$	66.7 msec
generate 384 random bits	$=$	6.4 sec
system boot (w/ attack)	$=$	11.92 sec
TEA encr. rate (1 round)	$=$	47k bytes/sec
TEA encr. rate (3 rounds)	$=$	15.7k bytes/sec

The critical file and desired files used in this benchmark were each 30,000 bytes in length. Note that there are no disk writes needed in the system boot phase of the virus, but disk writes are needed to infect the system file and program files. This is why the system boot phase takes much less time. We were unable to get the same random generation rate that [LMS] got using truerand. We found that the Macintosh SE/30 can only generate 1 random bit per clock tick (1/60th of a second), as opposed to 2 bits per tick. The random number generation is the bottleneck in terms of CPU time, taking up 53.7% of the attack time. It takes a mere 4 ticks to encrypt the plaintext using RSA.

Table 2
Virus Size

code size	bytes	src language(s)
attack routines 'main'	434	ANSI C
TEA encryption routine	88	Asm
truerand size	124	Asm
misc. attack code	804	ANSI C
global data	560	N/A
modified GNU MP lib	4,372	ANSI C
entire attack routine	6,382	ANSI C/Asm
main virus routine	614	Asm
total virus size	6,996	ANSI C/Asm

It can be inferred from table 2 that the attacking routine could be made smaller if the entire routine were written in assembly language. One of the outcomes of our research was that we found that it is possible to write code for RSA, truerand, and TEA, such that the code does not exceed 7k bytes. Optimizing the code for size was a major challenge since most viruses are very small in size. The only limitations that were placed on our code is that it contains a public key with a small exponent built into it, along with the inverse of the composite modulus. These optimizations allowed us to omit a multiprecision exponentiation routine and a division routine. This may have applications in other areas such as smart card technology.

For space comparison, note that we used a modified GNU MP library which comprised only 4,372 bytes. The size of the object code for the GNU library that is required for full RSA encryption and decryption for the Macintosh is 14,818 bytes. Note that this object code corresponds to source files that are *entirely* in ANSI C. Our virus has in-line assembly, miscellaneous optimizations, no exponentiation code, and no division code. This is what accounts for the big difference in size. The value of 14,818 does not include the C standard library code (which is about 27k itself).