CAN CRYPTOGRAPHY PREVENT COMPUTER VIRUSES?

John F Morar & David M Chess
IBM TJ Watson Research Centre, PO Box 704, Yorktown Heights, NY 10598, USA
Tel +1 914 7847368 • Fax +1 914 7846054 • Email morar@us.ibm.com, chess@us.ibm.com

ABSTRACT

The relationship between cryptography and virus prevention is anything but simple. Since the beginning of the computer virus problem, people have proposed solutions involving some form of cryptography; but cryptography plays only a minor role in the solutions we actually use today. Encryption can also make virus prevention more difficult, by providing viral hiding places inside the objects that it protects. This paper will provide an overview of the ways that encryption technology impinges on virus protection and related security efforts, and provide some understanding of how encryption can help, or hurt, the efforts of the 'good guys'.

CRYPTOGRAPHY EXPLAINED

For centuries, cryptography has been used to keep secrets. In traditional, symmetric, single-key cryptography a message (the 'plaintext') is transformed, using a key, into another form (the 'ciphertext') from which it cannot be recovered without knowing the key (or, in reality, from which it is very difficult to recover without knowing the key). Two people who both know the key can communicate securely even through an insecure channel as long as the key is kept secret; an attacker intercepting a ciphertext message cannot determine its plaintext content, lacking the key. Converting a plaintext message into the corresponding ciphertext is called 'encryption'. Converting ciphertext into plaintext by use of the key is 'decryption'. Converting ciphertext into plaintext *without* using the key is part of 'cryptanalysis', the science of code-breaking.

A related use of cryptography is the production of modification-detection codes, also known as cryptographic checksums or cryptographic hashes. A modification-detection code is a small number (typically between 16 and 128 bits long) which is derived by an algorithm from a large dataset, in such a way that it is very difficult to find another, different dataset for which the algorithm produces the same small number. One important use of modification-detection codes, as suggested by the name, is to determine whether or not a file has changed, without having to maintain a complete copy of the file for later comparison. By storing only the much smaller modification-detection code corresponding to the original state of the file, it is possible to verify (with high probability) that the file is unchanged at a later time, by re-executing the algorithm and verifying that the result is the same as the stored value. Verifying that a dataset has not changed is often referred to as verifying the 'integrity' of that data.

In recent years, asymmetric cryptographic algorithms have appeared, in which different keys are used for encryption and decryption, and someone knowing only the decryption key cannot feasibly determine the encryption key. This has made digital signature technology possible: if I generate a pair of keys, keep the encryption key to myself, and reveal the decryption key to the world, I can now produce a message, encrypt it with the secret encryption key, and publish it. Others can verify that the message was indeed produced by me (or at least by someone who knows my secret encryption key), by using the publicly available decryption key to decrypt the message. (In practice, asymmetric cryptographic algorithms are very slow, and real digital signature systems usually involve computing a cryptographic hash of the message and encrypting that smaller piece of data using the asymmetric encryption key; but that is a detail.)

For a good, and often amusing, survey of modern cryptographic algorithms and how they are used in various popular communication protocols, see endnote 1. The best-known asymmetric cryptosystem, used for both secrecy and digital signing, is probably PGP, described in endnote 2.

As the 'Net and the Web move into more central positions in the life of the world, the functions that cryptography provides (including secrecy, integrity, and digital signatures) become more important, and cryptographic functions can be found in more places, doing more things. Bruce Schneier writes:[3]

> *'From email to cellular communications, from secure Web access to digital cash, cryptography is an essential part of today's information systems. Cryptography helps provide accountability, fairness, accuracy, and confidentiality. It can prevent fraud in electronic commerce and assure the validity of financial transactions. It can prove your identity or protect your anonymity. It can*

> *keep vandals from altering your Web page and prevent industrial competitors from reading your confidential documents. And in the future, as commerce and communications continue to move to computer networks, cryptography will become more and more vital.'*

Cryptographic functions have always been present in computer systems, but they have usually been relegated to a few obscure utility programs, function calls, or extra cost add-ons. We are now starting to see rich cryptographic functions incorporated into user operating systems and widely deployed applications. What are the implications of the increasing power and ubiquity of cryptography for the battle against computer viruses? How do viruses use cryptography, how do anti-virus programs use it, and what role does it play in the design of present and future security systems that can help us make our computers resistant to viruses and related threats? These are the questions we intend to address in the rest of this paper.

CRYPTOGRAPHY IN COMPUTER VIRUSES

Some computer virus authors have made use of cryptography themselves, in attempts to make their creations more difficult to detect or more difficult to analyse, or as part of the destructive or annoying payloads they carry. While cryptography does not play a key role in most of the viruses currently responsible for the virus problem, the issue of cryptography in computer viruses is worth at least a passing glance.

The most common use of encryption in computer viruses is as part of polymorphism. A polymorphic virus is one that changes form as it spreads, so that two different files infected with the virus will not necessarily have any significant byte-strings in common. In machine-language viruses, polymorphism is usually achieved by splitting the virus into three different sections: a short piece of decryptor code, a cryptographic key, and the main functional part of the virus. When the virus creates a new copy of itself, it selects a new cryptographic key, encrypts the main functional part of itself with that key, and generates (using any of a variety of methods) a new implementation of the decryptor code. When the virus executes, the decryptor code runs and uses the key to decrypt the main functional part, which then receives control. Since changing the key changes the bytes in the encrypted body, and because the decryptor code and the key vary with each copy of the virus, two different instances of the virus will contain very different byte-strings.

Note that this technique of encryption for polymorphism is using cryptography only to transform data, not to keep a true secret. Since the decryption key is stored within each instance of the virus, a virus analyst can always find the contents of the main virus body, simply by doing the decryption directly, using the stored key (and in fact this is the most common way that anti-virus programs detect this kind of polymorphic virus).

A few viruses use encryption in a more powerful way, by encrypting parts of themselves and *not* including the decryption key within the virus. Without the decryption key, a human analyst cannot determine what that part of the virus would do if it were to be decrypted and executed. Of course, without the decryption key the virus itself cannot decrypt the hidden code, either! To be effective, such a virus must have a way of eventually finding the key, and recognizing that it has found it. An early DOS virus, for instance, searched the filesystem for a filename having a certain checksum, and when that filename was found it was used as a decryption key to decrypt a certain piece of its own code, which was then executed. Since the encryption was weak and the

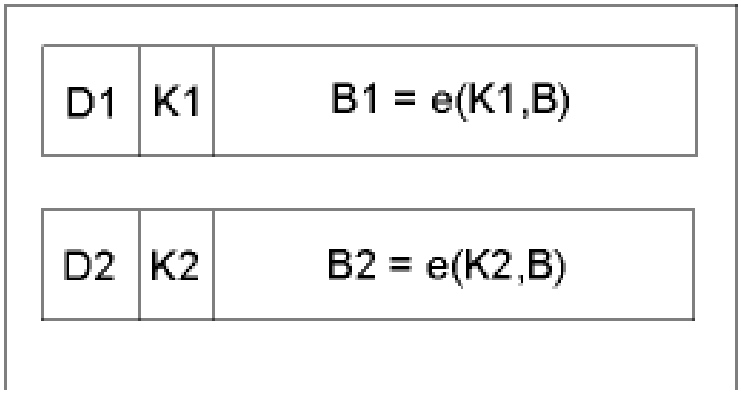

Figure 1: Two instances of a polymorphic virus that uses encryption. In the first instance, decryptor D1 uses key K1 to decrypt B1, which is the main body of the virus B encrypted with K1. In the second instance, decryptor D2 uses K2 to decrypt B2, revealing the same B. Although both instances eventually execute B, they may have no significant byte-strings in common.

checksum algorithm easy to reverse-engineer, anti-virus workers were able to determine that the filename used as the key was associated with a particular Bulletin Board program, and the encrypted code would alter the program's configuration files to introduce a security hole. This sort of encryption can, in theory, make parts of a virus opaque to analysis; in practice, however, the technique has not had any significant impact on the world. (See endnote 4 for a theoretical treatment of the technology.)

A more significant use of encryption in viral payloads involves using encryption *as* payload. Some *MS Word* macro viruses, for instance, will, on a certain date or with a certain probability, save the infected document with a password that is not known to the user. This is similar in effect to simply erasing the file (since the user no longer has access to its contents), but is psychologically more frustrating, since the user has the feeling that their data is all still there, if only the password could be found. Some viruses use a fixed password (so the user can in fact recover the data after reading a sufficiently detailed description of the virus), while others use a randomly generated password (so recovering the data requires cryptanalysis of the encrypted document; this cryptanalysis is in fact often possible in earlier versions of *Word*, but more modern versions use stronger encryption). In machine-language viruses, the One_Half virus gradually encrypts parts of the hard drive on an infected machine, and while the virus is in memory it dynamically decrypts any encrypted sectors that are read. Removing the virus from the hard disk without undoing the encryption can result in a computer that is disinfected, but whose hard disk contains some garbled data. For thoughts on similar uses of cryptography in viruses and malicious code, see endnote 5.

The last use of encryption by malicious software that we will mention involves Remote Access Trojan horse programs, sometimes called 'backdoors'. A Remote Access Trojan horse is a non-viral malicious program which, once the user has been tricked into executing it or the attacker has otherwise arranged for it to be installed on the target machine, listens to the network for queries and commands from the attacker. Some of these Trojan horses use encryption on the communication channel between the attacker and the listening program, for the traditional reasons of secrecy (see for instance David Dittrich's analysis of the 'Stacheldraht' program[6]). When the communication channel is encrypted, it is more difficult automatically to recognize traffic flowing between an attacker and a compromised host.

CRYPTOGRAPHY AND VIRUS PREVENTION

Cryptography is difficult to get right. As Schneier points out[3], even if I have a perfect cryptographic algorithm that no one alive can beat, turning that algorithm into a working system involves multiple layers of architecture, design, coding, and user interface, and a mistake anywhere along the way can render the resulting system completely insecure, despite the soundness of the basic algorithm. As we analyse the usefulness of various cryptographic techniques in the prevention of computer viruses, we will generally assume that all the hard work has been done right: that the systems are not only based on sound cryptographic algorithms, but are correctly and securely designed and implemented. Needless to say, if that is not true and the systems are in fact insecure, their usefulness in virus prevention will be significantly reduced!

Cryptography as a roadblock

We will first consider a class of cases in which cryptography is a barrier to effective virus prevention. As outlined[7], there are a number of situations in which encryption of potentially infected data prevents that data from being examined for the presence of viruses. In particular, whenever encryption has been used to restrict the ability to read a dataset to some set of entities, and the entity attempting to check the dataset for viruses is not in that set, the encryption will prevent the virus check. Some of the most common cases of this are:

- a virus scanner in a network gateway or firewall unable to check encrypted traffic;
- a virus scanner in a mail gateway unable to check encrypted mail for infected attachments;
- a virus scanner in a mail gateway unable to check encrypted attachments for infection;
- a virus scanner on a file server unable to check encrypted files stored on the server;
- a virus scanner on a client machine unable to check encrypted files stored locally.

There are a number of measures available to address this problem.

Users making use of encryption may be required to have up-to-date, real-time virus scanning in place on the client machine, where it has the best chance of seeing the ‘to-be-encrypted’ objects while they are still plaintext. It is notoriously difficult, especially with the proliferation of mobile users and laptop computers, to track all the client systems that might be attached to your enterprise Intranet and ensure that certain software is always installed and active, although various commercially-available enterprise anti-virus solutions take some steps in that direction. On the other hand, having good anti-virus software on client systems is desirable in any case, so most enterprises already have such a requirement, however well or badly they are able to enforce it.

Cryptographic facilities in the enterprise may be configured to always include a particular enterprise administrative key when encrypting a dataset, and virus-scanning processes may be given that key. However, not all cryptographic facilities include such a feature (the *Windows 2000* recovery key has a similar but not directly applicable function, and recent enterprise versions of PGP apparently have a related feature), and the risk posed by the possible theft of that adminis-

trative ‘read everything’ key may be too high to bear.

End-to-end encryption may simply be forbidden in an enterprise – with all traffic checked for viruses before it leaves the trusted Intranet, and encrypted afterward for travel outside (via, for instance, a Virtual Private Network). However, enforcing such a ban will, in practice, be difficult.

It is worth noting here that access control methods besides encryption have similar problems; in *Microsoft Windows NT*, for instance, it is possible to set a file’s permissions so that not even an administrator can read the file in the normal way. However, users generally want such files to be backed up, and administrators want such files to be scanned for viruses. So the operating system file-access routines include the notion of ‘backup semantics’, which allow accessing even those files which are normally not readable except by a particular user, and most commercial anti-virus programs use this feature in virus checking. The tradeoff here is similar: it is desirable to let certain processes (backup programs, virus scanners) access even data that is nominally readable only by a small set of users, but doing so has certain security implications; if an administrative account that can use backup semantics is compromised, or if the enterprise administrative key is stolen, an attacker can access everything in the system, just as the virus scanner can.

Cryptography as a resource

We have seen that cryptography can be used by viruses, and can interfere with virus prevention measures, but surely we can also derive some *benefit* from cryptographic techniques in our efforts to prevent computer viruses?

A virus cannot infect what it cannot see. This would suggest that encrypting our programs and our program-carrying objects (such as *Microsoft Office* documents) might help protect them against viruses: if a virus cannot read the plaintext of a file, it will presumably not be able to alter the file in order to infect it. Facilities to automatically encrypt everything written to disk have been available in various forms for many years. With *Windows 2000*, *Microsoft* now offers a simple ‘point-and-click’ method of encrypting some or all of the filesystem (see Figure 2).

In practice, though, the most successful viruses infect objects that are actually in use (programs that are being executed, documents that are being edited), and those objects must be present in

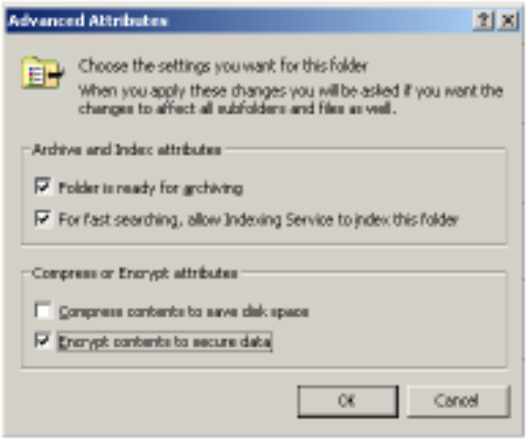

Figure 2: User interface to Windows 2000 file encryption

decrypted form. If the user is able to access the plaintext of a file to work with it, a virus running with the user's privileges is likely to be able to access it to infect it. So encryption of files in the filesystem has not been, so far, an important part of virus prevention.

Encryption is traditionally used on communication channels, to prevent anyone intercepting traffic on the channel from reading the contents of messages. Encryption of communication should similarly prevent programs and program-carrying objects from becoming infected in transit. Infection-in-transit has never been a significant part of the virus problem; secrecy, rather than virus protection, is by far the strongest reason to encrypt communications.

If encryption for secrecy is not a promising resource in the battle against viruses, perhaps we can get some mileage out of modification detection codes. As long ago as 1987, Pozzo and Gray[8] proposed a system in which a cryptographic checksum is used to verify that a program has not been changed since it was installed, and to forbid execution of any program that fails that test. No such system is currently in widespread use. This may be simply because the makers of the few widespread virus-bedevilled operating systems have not chosen to implement the function; on the other hand, actually implementing such a system involves solving some tricky problems. In the home environment, there must be a simple and convenient mechanism whereby an unskilled user can install a new program, or upgrade an existing program; then again, that mechanism must somehow be protected so that a virus cannot use it to install itself. In the enterprise environment, while there may be a more skilled IT department competent to bring in and 'bless' new programs, the process of distributing these new programs while again preventing viruses from similarly distributing themselves, presents a challenge.

Anti-virus programs have made more limited use of modification detection codes in detecting modifications to the anti-virus code itself. Most anti-virus programs will detect, at some level, when the anti-virus code itself has been altered, and will warn the user and in some cases refuse to run. This is useful, of course, only when the anti-virus program contains or consists of files of the same type that the active virus infects. An anti-virus program that consists of binary executables will never detect in this way a virus that only infects *Office* documents. Anti-virus self-checking is a useful precaution, but it is not a major weapon in the battle. Similarly, *Windows 2000* contains a System File Checker, which apparently uses a modification detection code to detect changes to certain operating system files, and restore the original from a backup or CD-ROM when a change occurs. This function does deter certain viruses, but that was not its primary purpose, and since it monitors only a small fraction of the executables on a typical computer, it provides only a little protection.

Close in 'idea-space' to forbidding any program from running if it has been modified, is forbidding any program from running unless it is on a list of programs that are known (or at least believed) to be uninfected. Cryptography is involved again, because the only feasible way of accomplishing this is to keep a list of the cryptographic checksums of each of the approved programs, and to allow execution of a program only if its checksum matches one in the list. We know of no currently-available security package that does this for programs of all types; again there are significant administrative issues involved in the creation and distribution of the approved list, as well as in preventing a virus from adding the checksum of an infected program to the list. But in the limited field of *Office* macros, there are at least two offerings that provide this function: both *Symantec's Macro Virus Protection*[9] and *F-Secure's Macro Control*[10] allow the user (or administrator) to prevent the execution of any office macro that is not on the approved list. While this should provide complete protection against macro viruses, neither product seems

to have been very successful in the marketplace. Either customers simply do not know what they really need, or the administrative overhead involved in creating and maintaining approved-macro lists outweighs the increased protection against viruses. (Both products come with a list of common, clean macros known to the manufacturer but, of course, many enterprises have their own extensive libraries of macros that would have to be added.)

If deciding which programs should be allowed to run on a case-by-case basis is too much work, perhaps we can simplify things using digital signature technology. Can we prevent viruses by only accepting programs (or program-carrying objects) from people that we trust? Anti-virus software (such as the *Digital Immune System* from *Symantec*) has begun to use digital signatures to ensure that what seems to be an anti-virus update is actually coming from a trusted source; can we use this ability in the wider anti-virus arena?

Microsoft Word 2000 contains a set of features that allow the macros in an *Office* document to be signed, and allows a user to specify that no macros should be allowed to run unless they are signed by someone who appears on a list of trusted signers (see Figure 3).

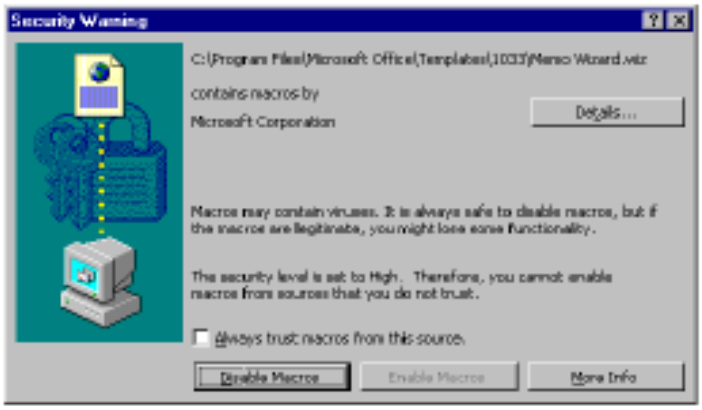

Figure 3: Microsoft Office 2000 macro security warning box

Again, the idea here seems promising, but it is not clear how widely used or effective it will be in practice. See endnotes 11 for a brief analysis of some of the limitations of the *Office 2000* macro-signing features. In particular, if you elect to trust one or more people who are not sufficiently careful themselves and become infected with a macro virus, nothing in the macro-signing system will prevent at least the payload of the virus from executing on your system. This is a general problem with the idea of judging the cleanliness or trustworthiness of a program by the identity of the agency signing it: if that agency makes just one mistake, there will be a malicious piece of software out there with the agency's signature on it, and unless the signature infrastructure allows by-object revocation ('if you receive the object with checksum X, treat it as unsigned even though it appears to be signed by A'), the only alternative is to revoke the agency's signing key entirely and start over from scratch, redistributing a new set of objects signed by the so far mistake-free key.

Deciding who to trust is clearly important in any system based on digital signatures. Left to themselves, individual users will make poor trust decisions now and then. When presented with a dialog whose options are essentially 'Continue with what you were doing by trusting person X' or 'Cause the current operation to fail', users almost invariably choose the former, because it lets them get on with whatever they were doing, and because in practice it only rarely has negative

consequences. Still, such casual trust decisions happening all over a company can be expected to quickly dilute whatever security the signature scheme originally offered. To avoid this, administrators will want to have control over the trust-lists of their users. There are unsolved problems here, though, as well.

If only a few signers are trusted, then those signers will quickly become the bottleneck in application development; either they will be overloaded with new programs to sign, and therefore slow down the development process, or people who might have taken advantage of the programmability of modern systems will be put off by the signing process, and continue to do things in inefficient manual ways rather than writing new programs or macros. On the other hand, if many signers are trusted, the odds that one of them will be careless and become infected increases, and of course such an infection will be able to spread despite the security system.

The *Office 2000* macro-signing system, as well as the security offered by *Microsoft* ActiveX, depend on an all-or-nothing approach to security. If you give a macro or ActiveX control permission to run at all, it runs with your full privileges, and can do anything that any program run by you can do. The same applies to a script received as an attachment to a mail message; if you allow it to run at all rather than simply deleting it, it can do anything that you can do, including erasing your files, sending hundreds of copies of itself in email under your name, and so on.

Rather than allowing any program that runs to do anything it wants to do, might we derive some advantage from limiting in a more fine-grained way what some programs can do? Can we use cryptography-based digital signatures to determine who vouches for a program, and then use that to determine what it should be allowed to do? Perhaps most programs do not require any but the most benign functions, and therefore do not need to be signed, and the programs that do require special privileges and powers will be few enough that they will not overload a central signing authority.

Stepping back for a moment to look at the problem from afar, this seems like just what we want. Viruses and other Trojan horses work by exploiting trust relationships. In a discretionary access control system (which the vast majority of security systems currently in use are), I am permitted to read certain files because I am trusted not to abuse or wantonly distribute the information I find there. I am permitted to write to certain files because I am trusted not to implant any dangerous or destructive code there. But of course, within the computer I cannot do anything directly myself, I can only run programs. If one of those programs is malicious, and is able to run with my full privileges, it can exploit the trust that has been placed in me. It can steal secrets that I would never steal, by executing *as* me. It can plant copies of itself in programs that I would never sabotage, by doing so *as* me. If we can break the ability, in at least most circumstances, for malicious code to run as though it were me, we should be able to go at least some way toward preventing viruses and related threats.

So are there any systems that allow this kind of granular program control available today?

Both *Lotus Notes* and *Sun's* Java allow for a considerable degree of this kind of control (see Figures 4 and 5).

In *Lotus Notes*, each client system is governed by an execution control list, which specifies which kinds of activities are allowed to scripts and agents signed by which signers. The example in Figure 4 shows that for the default signer (that is, for scripts signed by signers not explicitly listed in the execution control list), no actions whatever are allowed by default. *Notes* also allows

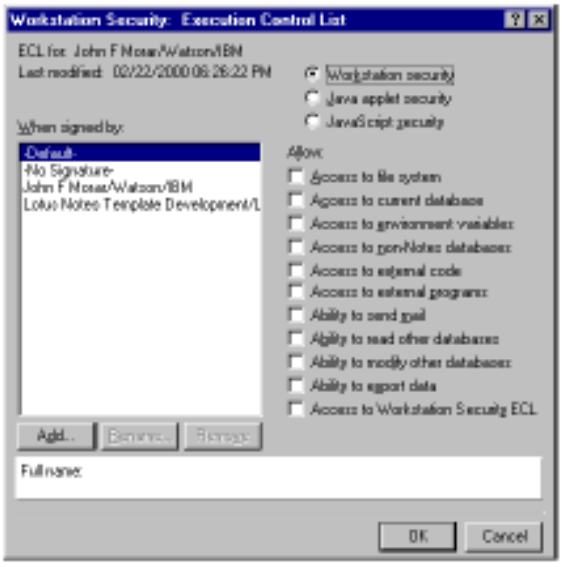

Figure 4: Lotus Notes Execution Control List (ECL) dialog

the administrator to specify whether or not a user should be able to allow an action that is not allowed by default; if the administrator turns on this ability, the user will get a dialog asking if the action should be allowed this once, or if this signer should be added to the execution control list as permitted to do all actions of this type. See endnote 12 for more details on the granular security features in *Notes*. Java 2, the most recent version of *Sun's* Java family of products, offers similar granular execution control. A user can specify, for each kind of activity that the Java runtime environment controls, which Java programs should be allowed to carry out those activities; the programs can be specified by signer, by codebase (i.e. by where on the Web the program is loaded from), or both (see Figure 5).

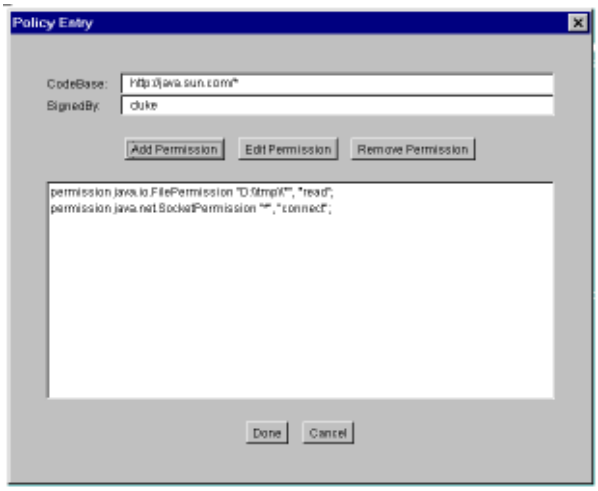

Figure 5: The Java Policy Tool – adding a permission entry

Since there are no native *Lotus Notes* or Java viruses active in the wild, it is hard to judge how well this sort of granular execution control actually does in preventing viruses. There are difficult problems that the current systems have just begun to address; again, users will tend to be impatient with complex configuration tasks (or even simple configuration tasks), and will tend to

press whichever button seems most likely to let them get on with their current activity. The original Java 2 implementation required users to use the policy tool (illustrated in Figure 5), or to manually edit a security policy file in order to authorize programs signed by a new signer to take otherwise-prohibited actions on their systems. It is suggestive that in a later release of Java 2, *Sun* added an additional security policy which bypasses all the complexity and power of the granular system, and simply asks the user at runtime 'should this program signed by this signer by given full run of the machine?'.

The fact that *Sun* felt the need to introduce this very simple, but arguably much less secure, path into the system suggests strongly that we still do not know how to make a system that is sufficiently trivial to use that users will actually tolerate it, while at the same time not offering a simple 'bypass security entirely?' button for them to push.

We believe that the technology of granular execution control, backed by strong cryptographic digital signatures, will go a long way toward increasing the security of our systems in general, and that such security is vitally needed in a world where code is increasingly mobile and increasingly likely to reach systems other than the one on which it was written. If we had had functioning granular execution control the LoveLetter virus would not have spread, because a piece of code signed by a stranger in the Phillippines would not have been allowed to send out hundreds of copies of itself in electronic mail.

Similarly, a Win32 Trojan horse arriving at a victim's system would fail to install, since a random program signed by a stranger, or not signed at all, would not be allowed to alter the Registry, or install itself in the TCP/IP stack. On the other hand, a granular execution control system in which the majority of users have at some point in the past pushed the 'trust absolutely everyone to do absolutely everything' button, just to get rid of those annoying security pop-ups, will do us little good at all. Finding a way to do the former without falling into the latter is a challenge that we are just beginning to face.

Note that we do not mean to imply that granular execution control, if we figure out how to do it right, will completely eliminate the virus problem. Sometimes I will still unknowingly run a malicious program with just a little too much privilege, and it will begin to spread as viruses always have along the existing lines of trust. We will still need known-virus scanning, and we will still need an immune system. However, complementing those facilities, we believe that granular execution control systems built on cryptographic signatures will play an important role in making our systems secure into the future.

CONCLUSION

We have briefly surveyed the uses of cryptography in viruses, in anti-virus software, and in general security systems as they apply to viruses and related threats. While cryptographic techniques alone are only tools, far removed from the finished working systems that we need to build, they are potentially useful tools. In some cases cryptography can actually make virus protection more difficult; we have tried to outline those cases and the various methods that can be used to overcome that difficulty. In some cases cryptography is irrelevant to virus protection. However, in some cases, digital signatures in particular, we believe that cryptography will play an important role in the way our systems are secured in the future, both against viruses and against the more general class of emerging threats.

ENDNOTES

1 Charlie Kaufman, Radia Perlman, and Mike Speciner, *Network Security: Private Communication in a Public World*, Prentice-Hall, 1995.

2 Simson Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly, 1994.

3 Bruce Schneier, 'Why Cryptography is Harder Than it Looks, http://www.counterpane.com/whycrypto.html

4 James Riordan and Bruce Schneier, 'Environmental Key Generation Towards Clueless Agents', in G Vinga (Ed), Mobile Agents and Security Springer-Verlag, *Lecture Notes in Computer Science* No.1419, 1998.

5 Adam Young and Moti Yung, 'Cryptovirology: Extortion-Based Security Threats and Countermeasures', Proceedings of the 1996 *IEEE* Symposium on Security and Privacy, May 6–8, IEEE Computer Society Press, 1996, pp.129–140.

6 David Dittrich, 'The Stacheldraht distributed denial of service attack tool', 31 December 1999, http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt

7 David Aubrey-Jones, 'Combining Encryption with an Anti-Virus Strategy', Proceedings of the Eighth International *Virus Bulletin* Conference, October 1999, pp.205–234.

8 Maria Pozzo and Terence Gray, 'An Approach to Containing Computer Viruses', *Computers and Security v6n4*, 1987, pp.321–331.

9 http://www.symantec.com/education/var/modules/m5tab5i.html

10 http://www.Europe.F-Secure.com/news/1998/19980316.htm

11 Paul Ducklin and Philip Hannay, 'Microsoft Office 2000 and digital macro signatures, revision of April 2000, http://www.sophos.com/virusinfo/whitepapers/office2000.html

12 Martin Overton, 'Viruses and Lotus Notes: Have Virus Writers Finally Met Their Match?', Proceedings of the Ninth International *Virus Bulletin* Conference, 1999, pp.149–174.