Trusted Distribution of Software Over the Internet

appeared in the Internet Society 1995 Symposium on Network and Distributed System Security

Aviel D. Rubin
Bellcore
445 South St.
Morristown, NJ 07960

Abstract

This paper offers a solution to a problem of software distribution on the Internet. The problem is that malicious software can be posted to the public with no accountability. When this software is run, it inherits the privileges of the user who runs it. Unfortunately, it is very common for users to execute software obtained on the Internet with no assurance that it is genuine.

The solution offered here utilizes a trusted third party that signs certificates to identify the author of a program and to secure its integrity. A detailed design is provided. Finally, Bellcore's Trusted Software Integrity (Betsi) System, *an implementation of the design, is presented.*

1 Introduction

The Internet is a dangerous place to live. Consider that there are thousands of people with unprecedented computational resources and unbounded time and patience making every effort to attack the estimated 2.2 million hosts on the Internet [3]. The susceptibility to attack is a major drawback to the wealth of services the Internet affords. However, most users seem willing to take their chances.

One of the most useful features of the Internet is that millions of users can share data, programs, pictures, and other information with people they have never even met. For many people, it is the primary means for sharing information. The trend to use the Internet as a means of transporting electronic data will grow as the information super-highway becomes a reality.

Many software vendors distribute updates to their software through the Internet. Also, popular programs are circulated by advertising an *ftp* site. For example, the developer of a program can make it available to the public with an announcement in a newsgroup. Figure 1 is an example of such an announcement. Mr. Romario has given instructions on how to obtain a piece of software using *ftp*. There are many users who obtain and run such software on a regular basis. In fact, it is not unheard of for a system administrator to obtain patches for the system binaries from a public *ftp* site and install them as *root*. The author is familiar with a number of sites that obtained the software for their Internet firewall in this manner.

Unfortunately, although it is common practice to obtain software from public sites on the Internet, it is also very dangerous. In Unix®, any program that is executed will run as the user who invoked it, with all of his privileges. So, if a user downloads a program, such as *run_me* in Figure 1, that program has access to all of his files. It can also send mail, post to newsgroups, and attempt illegal break-ins, all on behalf of the unsuspecting user. If *run_me* contains the following Unix command:

```
unix% find / -name \* -exec cat {} \; | mail \
Bebeto@company.com
```

then all of the files that the user can read, in the entire file system, will be mailed to Bebeto@company.com when the program is run. A slightly longer program could be written that would add a false *telnet* program in the user's home directory that mails off the user's login id and password, echos "Login incorrect", destroys itself, and then executes the correct *telnet*. These are just two simple examples of malicious programs that a user is vulnerable to if he downloads programs from the Internet and runs them. More sophisticated programs could do serious damage. It is left as an exercise to the reader to imagine the damage that could be done if a system manager installs software from a public site that will run as *root*.

PC users are at risk as well. It is easy for malicious users to insert viruses into a program that is posted to the Internet, and it happens all the time. A recent incident involves a posting to one of the `alt.binary.pictures` newsgroups where a PC virus was found in a shell archive. Little can be done to protect a personal computer that runs arbitrary software found on the Internet.

Posting to a newsgroup is not the only way to distribute malicious programs. There are central distribution points for many software packages that make very desirable targets for foul play. For example, the source code for the popular *Kerberos* authentication system can be found at `aeneas.mit.edu` in the directory `/pub/ATHENA/kerberos`. If crackers[1] manage to break in and replace the software with their own ver-

Unix is a registered trademark of Unix Systems Laboratories.

[1] A *cracker* refers to a malicious user who dedicates time and effort towards breaking into computer systems and causing harm.

```
Newsgroups: sci.crypt,alt.security,comp.security.misc
Path: tree!rutgers!korielbook!stem!news.secure.COM!oops
From: Bebeto@company.com (Bebeto Romario)
Subject: New Program Is Now Available....FREE!!!!!!!
Message-ID: <Bebeto4uIP.Lpm@company.com>
Organization: Free Software Development Company (555 555-1234 guest)
References: <4uIP$6lu@inapiate.School.EDU> <10234$43t@agate.univ.edu>
Date: Tue, 26 Jul 1994 17:06:18 GMT
Lines: 13

Announcing, a new FREE software package!!!!! This
is a new security package. Feel free to install it.
The package is meant to replace your DES encryption
library with a MUCH MORE SECURE version. It is alread
compiled and ready to go.

ftp to ftp.company.com in the directory
/pub/free-software/better-des/xxx/run_me

where xxx is the platform you will run it on. Once
you ftp the file run_me, just type: run_me at your
unix prompt, and your files will automatically be
updated.
```

Figure 1: **A typical posting to a popular newsgroup on the Internet.** This news posting is used to demonstrate how software is often distributed on the Internet. Typically, a user will announce an *ftp* site to which people can connect to acquire software, data, and documents.

sions, the user who downloads the software and runs it may be unaware that he is exposing his entire system.

In fact, it is not impossible for a cracker to cause a tampered version of a "trusted" program to be downloaded even without breaking into an *ftp* site by attacking the Domain Name Server (DNS) or hijacking the *ftp* connection [1]. Currently, there is no way for a user to verify that the program he possesses is the original.

There are even greater dangers when software is distributed through one of the newsgroups on the Internet such as `comp.sources.unix` and `comp.sources.x`. There are over 20 such newsgroups in widespread use. Various programs are posted in convenient formats. The most common is an executable shell script. The user types: *sh filename* and the file is automatically converted to some useful format. It is very easy for a cracker to post a shell program that many Internet users will execute without reading the source.

This paper presents a solution to the problem of trusted distribution of software on the Internet. The solution relies on the strength of cryptographic hash functions and digital signatures to guarantee that a file has not been modified from its original form. A trusted third party issues certificates that are used to verify the integrity of a file and its origin with some degree of confidence.

2 Previous work

The X.509 extended file system [7] provides a mechanisms for users to *endorse* a statement about a file with a digital signature. Thus, the author may want to provide a statement such as "This is as I wrote it." Another endorsement might indicate that the endorser has checked the file for malicious code. The endorsement is a signature of a digest of the statement made by the endorser.

As stated by the author of the X.509 extended file system, the end users will only want to trust a small number of endorsers, and so endorsement by people other than the author (of the file) are required. The X.509 extended file system provides attributes to files in a file system to check for integrity. The solution presented in this paper applies these concepts to the problem of integrity in software distribution on the Internet.

3 What is trust?

Trust is a difficult word to define. According to Webster, trust is "assured reliance on the character, ability, strength, or truth of someone or something [4]." Unfortunately, the words *assured reliance* are just as vague as *trust*. Trust is at the heart of the problem described above. If a user cannot *trust* a program to execute on his behalf, then he should not run it. The problem is how to obtain trust in a program.

Two things are required for a user to trust a program. He must trust the author of the program, and he must trust that the program he is executing is the same as the program written by the author. If a user does not trust the author of a program, then it does not matter if the program has been modified; the user will not run it anyway. Thus, the following definition of trust results:

Definition 1: *A user trusts the author of a program if, given a guarantee that he possesses the program in its original form, he is willing to execute it.*

A question now arises. If a user, U, trusts the author, A, of a program, does U also trust everyone that A trusts? Given the definition of trust above, this could be rephrased as, does U trust A to be competent in choosing which programs to run? This is an open question, and it depends on each individual user. The solution presented in this paper requires a trusted third party, T. However, trust in T is not based on whether or not someone would execute a program written by T. Rather, T must be trusted to tell the truth.[2] There are now two types of trust defined. The first (Definition 1) involves trust in running a program written by another person. The second involves trust in telling the truth.

The trust requirements for the solution presented in the paper are now given:

1. U trusts A as in Definition 1.
2. U trusts A's decision to trust someone to tell the truth.
3. A trusts T to tell the truth.

4 Assumptions

There are three types of entities on the Internet that are relevant to this discussion. There are authors, (referred to as A), who have some software to distribute. Users (referred to as U) wish to possess the software distributed by A. And finally, there is a trusted party, T, as described in Section 3. The following assumptions are necessary to use the technique for trusted distribution of software described in this paper:

- All parties can compute the same cryptographic checksum function.[3]
- All parties can verify digital signatures.
- A can generate a public/private key pair and sign with the private key.
- T can sign digitally.
- There is a well-known public key, ${K_T}^+$, associated with T.
- A has an identity that can be verified off line.

[2] In this context, truth means that when asked to perform a calculation, T performs it correctly and returns an accurate value.

[3] It is assumed that software to do this has already been distributed in some secure manner. This gets into a chicken or the egg problem discussed in Section 6.2.

5 Design

This section describes a solution to the problem based on the assumptions above. The first subsection describes the general idea; the details are deferred to the second subsection.

5.1 High level view

A typical scenario is given to provide a high level view of how trusted distribution of software occurs. In this scenario, an author, A, wishes to distribute files on the Internet, and T is a trusted third party as described in Section 3. The first process is called *registration*. A registers a public key, ${K_A}^+$ with T.

T obtains A's phone number by calling directory assistance, calls A, and verifies the information in A's registration message. The details are in Section 5.2. The out of band communication for A is over, and once registered, A signs future communication with ${K_A}^-$, the private key associated with ${K_A}^+$.

To distribute a file, A sends a signed message to T containing the hash of the file, $H(file)$ and other relevant information. Then, T issues a signed certificate containing the name of the file and its hash value, among other things. When A receives the certificate, he stores it along with the file. Finally, when a user, U retrieves a file through *ftp*, he also gets a copy of the certificate. The certificate is used by U to check the integrity of the file and the identity of its author.

5.2 The details

The previous section provided an overview of how software is distributed in a trusted manner. In this section, the specific details are given. Section 6 describes an actual implementation based on these design details.

5.2.1 Author registration

To register with T, an author, A, creates a message containing the following information:[4]

Name
Organization
E-mail address
City, State
Work phone (area code + no.)
Home phone (area code + no.)
Public key

A sends the message to T. After T receives the message, he calls directory assistance in the city provided in the message and obtains the home phone number of A. T can also obtain the work number by asking directory assistance for the number of the organization and calling the organization for A's work number. The important point is that the number is obtained off line.

Next, T calls A and verifies all of the information in the message, including the fingerprint of the public key [5] (which both A and T must compute). When

[4] The phone numbers in the message are just for comparison after they are discovered off line.

[5] A *fingerprint* is a small hexadecimal value computed from the key with a one-way function for easy comparison by humans.

T is satisfied that the message is accurate, he stores the information associated with A in a database and registration is complete.

Once A is registered, there is no need for further off-line communication.

5.2.2 File distribution

An author, A, must be registered before distributing files. To distribute a file, A constructs the following message:

author name
file name
file location (*ftp* site, optional)
file version number
name of hash function used (e.g. MD5)
cryptographic hash of file
date

Now, A signs the message and sends it to T.

When T receives the message, he looks up the author, A, in his database and retrieves his public key. Then, T checks that the signature computes and that the date is current. If everything is okay, then T generates a certificate, containing the following:

identity of T
name of hash function used (e.g. MD5)
author's name
author's organization
author's e-mail address
file name
file location (optional)
file version number (optional)
cryptographic hash of file
date

The identity of T is needed so that users know which public key to use if there are other certifying authorities on the Internet. The file location is included so that a verifier can check that he obtained the file from a valid distribution point.

After generating the certificate, T signs it with his private key, and sends it to A. T then stores a copy of the certificate in a publicly accessible location.

A checks that all the information in the certificate is correct and verifies the signature. If everything checks out, then A stores the signed certificate along with the file in a publicly accessible place and advertises their location. If there is a problem with the certificate, then A and T must work it out and figure out why. This may lead to the detection of an intruder.

The next step is for a user, U, to access the file. U connects to the location of the file and retrieves the file and the certificate. He then does the following:

1. verifies that T's signature on the certificate is correct
2. computes the cryptographic hash of the file and checks that it matches the one in the certificate
3. verifies any other information that is of concern to him, such as location, version number, author name, etc.

If the hash value computed by U is the same as the one in the certificate, and T's signature is verified, then U is guaranteed that the file has not been modified since it was registered with T.

6 An implementation

This section discusses an implementation of the above design for trusted distribution of software on the Internet. Several of the design decisions are motivated by the need to make something available immediately on the Internet.

6.1 Issues for deployment on internet

The design presented in this paper is targeted towards actual deployment on the Internet. A system that is not used and is not widespread is of little value. There are several requirements to that end.

- Use as much existing infrastructure as possible. The public key system that is most ubiquitous is PGP™[8]. MD-5 [5] is the most readily available hash algorithm. The code exists in many places, and it is a de-facto standard.
- Minimize software requirements for end users.
- Minimize computational requirements and communication overhead.
- Minimize the load on the trusted third party.
- Keep it simple

This paper describes an implementation that meets the above requirements and is available for immediate use.

6.2 The chicken or the egg

One of the assumptions mentioned in Section 4 is that all parties can compute the same cryptographic checksum. Therefore, they must all run the same program. For this to happen, there must be some way to safely distribute some software. Similarly, all parties must have code to verify digital signatures. As trusted distribution of software is the goal and a requirement, there is a Catch-22.

MD-5 and PGP were chosen for hash computation and public key cryptography respectively because of their wide availability. The code for MD-5 appears in an *rfc* by Ronald Rivest [5], as well as in Schneier's *Applied Cryptography* [6]. PGP is available from various sites, and it is signed (albeit with PGP) by its author, Philip Zimmerman, whose key is widely available. See Appendix A for information on where to obtain PGP, and code for MD-5.

PGP is a trademark of Phil Zimmerman.

6.3 Bellcore's trusted software integrity system

The design in section 5 has been implemented by Bellcore as an experimental research prototype. It is called **B**ellcore's **T**rusted **S**oftware **I**ntegrity (Betsi) System. The assumptions are the same as in Section 4, and they are instantiated as follows:

- All parties use the MD-5 hash function for file integrity.
- PGP is used by all parties for digital signatures.
- There is a 1024 bit PGP key associated with Betsi.

Although MD5 is recommended, Betsi has the flexibility to incorporate other hash functions. The hash function used is included in the certificate so that the author of a file chooses what hash function to use. Betsi is not tied to any particular one.

6.3.1 WWW access

The easiest and most direct way to use Betsi is through the world wide web. The URL for Betsi is `http://info.bellcore.com/BETSI/betsi.html`. Everything needed to interact with Betsi can be found on this page. For users without access to the web, the following sections describe how to interact with Betsi via e-mail.

6.3.2 To obtain help

For help on using Betsi send mail to `certify@bellcore.com` with the subject, **help**. You will receive a reply message containing instructions on how to register and obtain certificates. In addition, a copy of Betsi's public key will be included. It is advisable to obtain copies of this key from several independent locations and make sure they match to minimize the risk of an imposter publishing a fake key.

6.3.3 To register with Betsi

To register with Betsi, send a message to `certify@bellcore.com`, with the subject, **register**, and containing the following information (numbers represent line numbers): 1. name, 2. organization, 3. e-mail address, 4. City, State, 5. Work phone number, area code + number (optional) 6. Home phone number area code + number (optional) 7. ASCII armored PGP public key.

Your phone number will be verified/discovered through directory assistance in your city, and you will be contacted by phone for verification of the rest of your information.

If your private key is compromised, you should immediately send another request to register, and when you are called, explain that you need to register a new public key. Your old public key will be removed from the Betsi database. Revocation is discussed in Section 6.3.7.

6.3.4 To obtain a certificate

To obtain a certificate for a file from Betsi, create a message containing the following information (numbers represent line numbers): 1. your name, 2. file location (*ftp* site, optional), 3. "MD5" 4. file name 5. MD-5 hash of file,[6] 6. file version number (optional). If there are more than one file, then include the hash, name, and optional version number on a separate line for each file. Sign this message with the private key corresponding to the public key you registered. Make sure that the message is cleartext, with an attached PGP signature. Next, send this message to `certify@bellcore.com` with the subject, **certify**. You will receive a signed certificate from Betsi. Verify the signature, and make the certificate available with the file.

Figure 2 is an example of a message for obtaining a certificate.

6.3.5 The public key of Betsi

The public key of Betsi will be made as widely available as possible. To ensure that the key is valid, calculate the *fingerprint* of the key. In PGP, this can be done with the command:

unix% pgp -kvc certify

The PGP commands are platform independent. This will print out a 32 digit hexadecimal number. Compare your result with the following fingerprint:
`5F 34 26 5F 2A 48 6B 07 90 C9 98 C5 32 C3 44 0C`
If they are the same, then you are in possession of the right key. The fingerprint will also be verified by phone during the registration process to guarantee that all authors possess the right public key.

Although the fingerprint is a very good way of verifying a public key, the actual key is included as well (see Appendix B).

6.3.6 The private key of Betsi

Betsi's private key is the most important secret in the system. Anyone possessing this key can create arbitrary certificates. Therefore, it is vital to protect the private key from unauthorized parties.

Another concern is the loss of the private key. Betsi's public key has been widely distributed, and if the private key is erased, then a new key pair must be generated. Then, all of the users must update their public key ring. This becomes especially cumbersome when the verification is built into applications such as mosaic. Steps have been taken to meet both requirements: securing the private key and ensuring that a backup exists.

The private key of Betsi is kept on a PC in a file that is protected by a long and complicated passphrase. When a signature is needed, the file to be signed is downloaded to the PC using em ftp. No file systems are ever mounted. The PC is then physically

[6] See Appendix A on obtaining the code for MD-5.

```
To: certify@bellcore.com
Subject: certify

-----BEGIN PGP SIGNED MESSAGE-----

Bebeto J. Romario
thumper.bellcore.com
MD5
2ee29998b334297dcc32102eb857fbc3  Fontmap Version 2.1
c3e5704be89c4d4807d5fec11b1752e6  Great.Program.zip
dc3a8031d65db8bb0ec3de046abe79b2  README
012ffa34521f97ae381fb5f11f837016  Trusted.Dist.tar.Z

-----BEGIN PGP SIGNATURE-----
Version: 2.7

iQB1AwUBLoGbvJti/eSkC5bZAQFudQL/W4u7zOE9yp8rTXsSQK2L9pfHIYNzOkRB
g4lNKTHzFXgbEdN8+DzpB2mKCYYTNw+k/qP/Hui0yaG0oeksNYnqJN1zzB0vWhgG
TyFzJqVop/QlYIlrItFr+LD1ebDBOYon
=UGQZ
-----END PGP SIGNATURE-----
```

Figure 2: **An example of a message to Betsi for obtaining a certificate.**

disconnected from the network, and the passphrase is entered. Then, after the file has been signed, the passphrase and the secret key are cleared from memory, the network connection is established again, and the signed file is transferred back to the network and sent to its proper destination.

Pieces of the backup copy of Betsi's private key are in the hands of two of the directors at Bellcore, and both are required to reconstruct it. One director is in charge of a disk containing the secret key, under the passphrase. This disk is in his safe deposit box at the bank. Each director has a sealed envelope containing parts of the passphrase and instructions on how to reconstruct it. Neither director has enough information to know the private key.

6.3.7 Revocation

One of the difficult problems in any public key system is revocation. An author's public key needs to be revoked if his private key is compromised. Also, it may be desirable to revoke a certificate if an author discovers a problem with the software he distributed, or if he wishes to publish a new version.

In a naive solution to distribution of software on the Internet, a trusted third party could sign keys for authors, and they could then include their signed key in a certificate signed by them. Thus, the trusted third party is issuing an infinite linkage between the public key and the user. Unfortunately, there is no public key system in widespread use that includes expiration dates in public keys.[7] In fact, expiration dates don't solve the problem because there is no way to predict when a key will need to be revoked, and in assigning expiration dates, there is a difficult tradeoff between convenience and security. Therefore, there is no way to control a public key once it has been signed. In Betsi, this problem is eliminated because a user's key is revoked by removing it from the database of authors.

Betsi also maintains a list of revoked certificates. This list is dated and then signed. Thus, any user can verify that a certificate has not been revoked as of the date in the revocation list. The revocation list can be accessed through the world wide web, and it can also be found at `thumper.bellcore.com` in `pub/certify`.

7 Analysis

Betsi provides a binding of a file to some information about the file. A certificate means that the identity of the author has been verified by Bellcore using directory assistance. The holder of a certificate is assured that the file has not changed since it was submitted to Betsi. However, there is no guarantee that the file is benign.

A malicious user, Z, can register with Betsi and spread software containing trojan horses and viruses. However, there is some accountability. To register with Betsi, Z must use a real identity that can be verified through the phone company. One way to do this would be to use the name of someone, and then answer their phone to verify the message. However, Z does not know when the call verification will occur.

[7] TIS/PEM [2] includes a validity period for certificates of public keys. However, it is not as widely used as PGP.

Alternatively, Z could open an account with the phone company under a false name for the sole purpose of registering with Betsi. Although the phone company does its own credit checking and verification of new users, a determined Z could pull it off. Thus, while the level of assurance of author identity provided by Betsi is not perfect, it is much higher than previously existed on the Internet.

Once a user is registered with Betsi, there is no way an intruder can obtain certificates on his behalf without compromising the secret key. If the secret key is compromised, then the user registers a new public key, and the old one is discarded.

8 Conclusions

This paper identifies a serious security problem with current distribution methods of software on the Internet. The problem is that anybody can post software to the Internet, with no accountability. This software may be malicious, and if run by the user, could cause excessive loss. Unfortunately, many users still run untrusted software from the Internet.

A solution is presented that involves a trusted third party that issues certificates. First, the trust requirements are identified. Then, a design is presented, where the author of a program must register a public key with the trusted server. The author's identity is verified off-line as part of the registration process. Then, to obtain a certificate for a file, an author signs a request, and the trusted server sends him a signed certificate. The author makes the certificate available with the file, and users can verify the author's identity.

Finally, Bellcore's Trusted Software Integrity (Betsi) System, an implementation of this design, is presented. This implementation is available on the Internet.

9 Future Work

There are several extensions to this work under consideration. In the near future, Betsi will support TIS/PEM users and other public key infrastructures. It is also likely that Betsi will offer a key signing service. Thus, a Betsi signature implies that the author's identity has been verified off-line by Bellcore. In addition, using Betsi as a time-stamping service is under consideration.

There is a difficult problem that has not been addressed in this paper. There is no way for a user to know that a given distribution includes a certificate. Thus, if an intruder wishes to distribute a malicious version of some software, he can simply remove the certificate, and the user may never know it existed. The current solution is for the author to advertise that the software must be accompanied by a Betsi certificate. We hope to provide a more elegant solution to this problem.

Acknowledgements

The author thanks Milton Anderson, Alireza Bahreman and Aileen Cheng for helpful comments. Also, discussion with Shoshana Loeb were helpful in identifying the problem.

References

[1] Steven M. Bellovin. Packets found on an internet. *Computer Communications Review*, 23(3), July 1993.

[2] Trusted Information Systems. TIS/PEM user's guide for version 7.0.1, 1994.

[3] Peter H. Lewis. Computer snoopers imperil Pentagon files. New York Times, July 21, 1994.

[4] Merriam-Webster. *Webster's third new international dictionary of the English language, unabridged.* Merriam-Webster Inc., 1993.

[5] R. Rivest. The md5 message digest algorithm. *RFC 1321*, April 1992.

[6] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., 1994.

[7] Robert K. Smart. The x.509 extended file system. *Symposium on Network and Distributed System Security*, pages 129–137, February 1994.

[8] P. Zimmerman. Pgp user's guide. December 4, 1992.

Appendix

A Where to obtain software

There is a mosaic home page for Betsi at http://info.bellcore.com/BETSI/betsi.html. There are many places where PGP and MD-5 can be obtained on the Internet for personal, non-commercial use. The latest version of public domain software is 2.6.2 which is distributed at `net-dist.mit.edu` under the directory `/pub/PGP`. All other users can obtain version 2.7 of PGP from ViaCrypt, email address viacrypt@acm.org. Release 2.6.2 contains a directory called *contrib* with a program called *md5sum* that is very convenient for calculating the MD-5 hash of a file, such as that of Figure 2. The command *md5sum* generates a list of hash values with filenames of all the files in the current directory.

A copy of *md5sum.c*, signed by Betsi, is also available at `thumper.bellcore.com` in the directory `/pub/certify/md5sum`. A copy of the public key of Betsi can also be found there.

B The public key of Betsi

---BEGIN PGP PUBLIC KEY BLOCK---
Version: 2.7

mQCNAi5I0LwAAAEEAJZi970w+Lb7onAmrnExWKrgUFbjJku29qVRlBY6/UtUH+fW
s7MtAEUKIhktJ0cDpE+5Tbi6Lev2RXmXhT1hEjwxSwVFOMJmOuMZxlj+586IKigC
vVjF+hCFKQWRXsleM/axVbpH+pNUmWcK6QMdBDFlzS/9pxdAiBPcEwSgd4ahAAUR
tBxCZXRzaSA8Y2VydGlmeUBiZWxsY29yZS5jb20+iQCVAgUQLmN7F2V5hLjHqWbd
AQHW8QP+ID/93P2OPfqyJKyRWi2YALzLXoRHpWR0ObLryWyP2NiBttw/i6OiVs4R
3EjPD+faZgPYAtLLOk3canN4CDgNrkdcrA5eirw+oJkh13A2tyQloZjeVE+Iqdim
H49HMhPkNmWpDJW0MQg//aNwceDMHPfZiVy432oJVJflh5T4eO6JAHUCBRAuSNES
m2L95KQLltkBAXM3AvwJWTonyOEe6QoqOuR1lflVkzC6+Czv5khVHx9j6+kQGzoR
tYBCmabh6RULgPpQ5gJHEBGlA3T0zVoT2pLPpcqRhCOz1SX/FEu1rMMC5g4CRHE7
eaesVra7pNz6pdxlqZ4=
=B+1o
---END PGP PUBLIC KEY BLOCK---

Fingerprint:

5F 34 26 5F 2A 48 6B 07
90 C9 98 C5 32 C3 44 0C