Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG

KAHIL JALLAD[1,4] JONATHAN KATZ[2,4] BRUCE SCHNEIER[3]

[1] The Eon Company
kajal@eoncompany.com
[2] Department of Computer Science, University of Maryland (College Park)
jkatz@cs.umd.edu
[3] Counterpane Internet Security, Inc.
schneier@counterpane.com
[4] Work done while at Columbia University

Abstract. We recently noted [6] that PGP and other e-mail encryption protocols are, in theory, highly vulnerable to chosen-ciphertext attacks in which the recipient of the e-mail acts as an unwitting "decryption oracle". We argued further that such attacks are quite feasible and therefore represent a serious concern. Here, we investigate these claims in more detail by attempting to implement the suggested attacks. On one hand, we are able to successfully implement the described attacks against PGP and GnuPG (two widely-used software packages) in a number of different settings. On the other hand, we show that the attacks largely fail when data is compressed before encryption.

Interestingly, the attacks are unsuccessful for largely fortuitous reasons; resistance to these attacks does not seem due to any conscious effort made to prevent them. Based on our work, we discuss those instances in which chosen-ciphertext attacks *do* indeed represent an important threat and hence must be taken into account in order to maintain confidentiality. We also recommend changes in the OpenPGP standard [3] to reduce the effectiveness of our attacks in these settings.

1 Introduction

Electronic mail (e-mail) has become an essential and ubiquitous communication tool. As such, users and businesses have become concerned with the privacy of their e-mail and have turned to both commercially- and freely-available e-mail encryption software to achieve confidentiality. Typical users of e-mail encryption software are not educated in good security practices; it is therefore important to design *robust* software whose security is not compromised even when the software is used in a naive manner.

It was recently noted [6] that, in principle, the secrecy provided by commonly-used e-mail encryption protocols can be completely violated by an adversary using a *chosen-ciphertext attack*[1] [7, 1]. Furthermore, it was claimed that the attack

[1] In such an attack, an adversary given a *challenge ciphertext* C attempts to determine the underlying plaintext $P = \mathcal{D}(C)$ by submitting different ciphertexts C' to a *decryption oracle* that returns $\mathcal{D}(C')$.

could be implemented easily. We explore these claims in more detail; in particular, we attempt to implement the described attacks against GnuPG (available at `http://www.gnupg.org`) and PGP (available at `http://www.pgpi.org`) and thereby ascertain whether the attacks do in fact represent a serious concern. Our findings may be summarized as follows:

- We have successfully implemented the attack against GnuPG and PGP when files or messages are sent *without compression*.
- If compressed files are sent (e.g., a .zip file is sent using PGP), the attack still works and may be used to recover the original data.
- On the other hand, compression done by the encryption software itself (when an uncompressed file is sent) causes the attack to fail. In the case of GnuPG (when compression is used), the attack fails only due to the presence of a message integrity check which is not explicitly required[2] as part of the OpenPGP specification [3]. Without the integrity check, the attack succeeds 100% of the time.
- Implementations which strictly follow the OpenPGP specification [3] are vulnerable to the attack *even when the message is compressed during encryption*. As it turns out, the actual implementations of PGP and GnuPG deviate from this specification in significant ways, thereby (fortuitously) foiling the attack.

Our results lead us to suggest a number of changes to the OpenPGP specification.

We review the arguments of [6] as to why chosen-ciphertext attacks might be feasible in the specific context of encrypted e-mail. Imagine a user who has configured his software to automatically decrypt any encrypted e-mails he receives. An adversary intercepts an encrypted message C sent to the user and wants to determine the contents P of this message. To do so, the adversary creates some new C' and sends it to the user; this message is then automatically decrypted by the user's computer and the user is presented with the corresponding message P'. To the user, P' appears to be garbled; the user therefore replies to the adversary with, for example, "What were you trying to send me?", but also *quotes the "garbled" message* P'. Thus, the user himself unwittingly acts as a decryption oracle for the adversary. Using information obtained in this way, the adversary may be able to determine the original message.

2 Overview

The term "PGP" is an all-encompassing name for several implementations of an email encryption program first written as freeware by Phil Zimmerman in 1991 [5, 8]. PGP 2.6.2 was the first widely available and widely ported version,

[2] At the time this is written, RFC 2440 is being revised; future drafts may require the presence of a message integrity check [2]. Note, however, that if full inter-operability with older versions of PGP is desired then a message with no integrity check (as opposed to an invalid one) will be accepted and the attack described here can proceed.

followed shortly thereafter by versions fixing previous bugs as well as "international" versions free from patent and export-law restrictions. There were several commercial PGP 4.x releases which were compatible with the 2.6.x versions. PGP 5.0 and subsequent releases were GUI based (the previous versions being command line only); note that PGP 2.x and PGP 5.x versions are considered incompatible. An IETF working group was later formed to standardize the PGP message format, resulting in RFC 2440 [3]. GnuPG is a free, open source, RFC 2440 compliant implementation. Most of the different versions of PGP, as well as GnuPG, have a fairly significant user base.

2.1 The Attack

We explicitly consider attacks on PGP 2.6.2 and GnuPG. We refer the reader to [5, 8, 3] for a more in-depth description of the protocols; here, we merely provide a high-level description necessary for a proper understanding of the attack. Specifically, the attack exploits the symmetric-key modes of encryption used; therefore, only this detail of the protocol is presented. Further details of the attack can be found in [6].

PGP messages are encapsulated in packets. A PGP packet has a header section and a data section. The header holds information such as the packet type and length, among other things. The data section contains the payload of the packet, which is dependent on the packet type. The specifics of the packet format are slightly different in various versions of PGP and in the OpenPGP specification, we focus here on PGP 2.6.2 packets.

Consider an e-mail message (or file) M. PGP encrypts the message as follows:

1. A random "session-key" K is generated, encrypted with the recipient's public key pk, and encapsulated in a public-key encrypted session key packet (PKESKP). The output can be represented as follows:
$$\langle \text{PKESKP HEADER}, \mathcal{E}_{pk}(K) \rangle.$$
2. Message M is encapsulated in a literal data packet (LDP), resulting in:
$$LDP = \langle \text{LP HEADER}, M \rangle.$$
3. The LDP is compressed using the DEFLATE algorithm [4], and becomes the payload of a compressed data packet (CDP):
$$CDP = \langle \text{CP HEADER}, \textsc{deflate}(LDP) \rangle.$$
4. The CDP is encrypted with a symmetric-key encryption algorithm (i.e., block cipher) and key K, using cipher feedback (CFB) mode (described below). This gives ciphertext $C_1, C_2, C_3, \ldots$. The ciphertext is encapsulated in a symmetrically encrypted data packet (SEDP) as follows:
$$\langle \text{SEDP HEADER}, C_1, C_2, C_3, \ldots \rangle.$$

5. The following message is sent to the recipient:

$$\langle \text{PKESKP HEADER}, \mathcal{E}_{\text{pk}}(K)\rangle\langle \text{SEDP HEADER}, C_1, C_2, C_3, \ldots\rangle.$$

The recipient, reversing the above steps, uses his private key to compute K; given K, the recipient can then determine the compressed message which is decompressed to return the original message M.

A mode of encryption is necessary in step 4 (above) in order to encrypt CDPs longer than a single block. GnuPG and PGP use a variation of CFB mode which we now describe. Before encryption, the plaintext (i.e., the CDP) is prepended by a 10-octet string. The first 8 octets are random, and the 9th and 10th octets are copies of the 7th and 8th octets. This prepended data serves both as a weak "integrity check" and as the initialization vector for the cipher. We denote the resulting text by $R_1, R_2, P_1, \ldots, P_k$ where R_1 represents the first 8 octets prepended to the CDP, R_2 represents the last 2 octets (the "key check octets") prepended to the CDP, and $P_1, P_2, \ldots, P_k$ represents the CDP itself. Encryption using a CFB-like mode of encryption then proceeds as follows:[3]

Encryption: Given $R_1, R_2, P_1, P_2, \ldots, P_k$, compute:
$C_1 = R_1 \oplus \mathcal{E}_K(0^{64})$
$C_2 = R_2 \oplus \mathcal{E}_K(C_1)_{[0,1]}$ (note: C_2 and R_2 are 2 bytes long)
$IV = C_{1[2-7]} \circ C_2$
$C_3 = P_1 \oplus \mathcal{E}_K(IV)$
for $i = 2$ to k:
    $C_{i+2} = P_i \oplus \mathcal{E}_K(C_{i+1})$
Output: $C_1, C_2, \ldots, C_{k+2}$

Decryption: Given ciphertext $C_1, C_2, \ldots, C_{k+2}, compute:$
$R_1 = C_1 \oplus \mathcal{E}_K(0^{64})$
$R_2 = C_2 \oplus \mathcal{E}_K(C_1)_{[0,1]}$ (note: C_2 and R_2 are 2 bytes long)
$IV = C_{1[2-7]} \circ C_2$
$P_1 = C_3 \oplus \mathcal{E}_K(IV)$
for $i = 2$ to k:
    $P_i = C_{i+2} \oplus \mathcal{E}_K(C_{i+1})$
if $(R_2 == R_{1[6,7]})$
    Output: $P_1, P_2, P_3 \ldots, P_k$
else
    Error

As described previously [6], a single-message chosen-ciphertext attack can seemingly be used to decrypt any given ciphertext (for simplicity, we omit the PGP headers that are attached to the messages; these will be discussed in more detail later). Given ciphertext $<\mathcal{E}_{pk}(K), C_1, \ldots, C_k>$, to obtain the value of plaintext block P_i $(i > 1)$ one does the following:

---

[3] $\mathcal{E}_K(\cdot)$ represents application of the block cipher using session-key K. The notation $B_{[0,1]}$ represents the first and second bytes of a block B, and the notation $B_{[2-7]}$ represents the third through the eighth bytes of a block B.

1. Choose a (random) 64-bit number r.
2. Submit the ciphertext: $\langle \mathcal{E}_{pk}(K), C_1, C_2, C_{i+1}, r \rangle$.
3. Receive back the decryption P_1', P_2', where $P_2' = r \oplus \mathcal{E}_K(C_{i+1})$.
4. Compute $P_i = P_2' \oplus r \oplus C_{i+2}$.

(A similar attack may be used to determine P_1.)

Other chosen ciphertext attacks are also possible. For example, submitting:

$$< \mathcal{E}_{\text{pk}}(K), C_1, C_2, C_3, r_1, \ldots, C_k, r_{k-2} >,$$

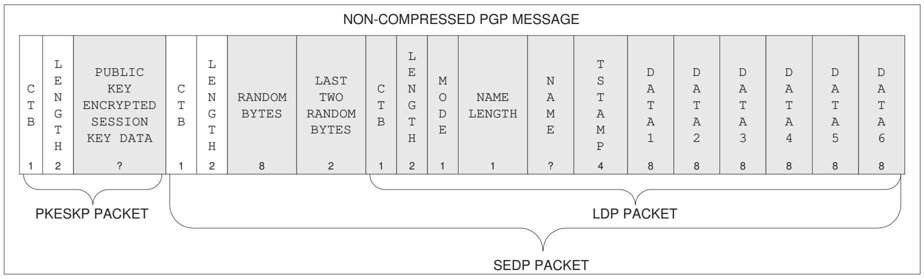
where $r_1, \ldots, r_{k-1}$ are random 64-bit strings, allows the adversary, in theory, to compute the entire contents of the original message.

As described, these attacks seem devastating to the secrecy of encrypted messages. In practice, however, one might expect certain complications to arise. We discuss this in the remainder of the paper.

3 Uncompressed Data

When the message data is not compressed by PGP before encryption (e.g., the compression option is turned off by the user), the encrypted message is simply an encrypted LDP. Note that if the original plaintext is already compressed (i.e., the plaintext is a zip file), it will be treated as literal data and will not be re-compressed by PGP.[4] We demonstrate here that the chosen-ciphertext attack as described in the previous section *does* succeed in recovering the plaintext message in these cases.

The diagram below represents a PGP message without compression. In the diagram, shaded portions represent encrypted data. The numbers along the bottom of the diagram represent the lengths (in bytes) of the corresponding fields. A "?" represents a field of variable length (for example, the length of the "Name" field depends on the value contained in the "Name Length" field). CTB is shorthand for *cipher type byte*, a PGP construct used to determine the type of the packet being processed. The "Name Length" field is a single byte that determines the length of the following "Name" field; this second field contains the name of the plaintext file. See [3] for more details of PGP packet formats.



[4] Although compressed files *are* re-compressed in GnuPG 1.0.6, this "flaw" was subsequently corrected in GnuPG 1.0.7 (thereby allowing the attack).

An attacker can follow the procedure given in Section 2.1 to produce a ciphertext that, when decrypted and returned, will allow the contents of the message to be determined. The diagram below represents the ciphertext that allows recovery of the entire message:

CHOSEN CIPHERTEXT MESSAGE (NON-COMPRESSED DATA)

CTB	LENGTH	PUBLIC KEY ENCRYPTED SESSION KEY DATA	CTB	LENGTH	RANDOM BYTES	LAST TWO RANDOM BYTES	CTB	LENGTH	MODE	NAME LENGTH	NAME	TSTAMP	R_1	DATA 1	R_2	DATA 2	R_3	DATA 3
1	2	?	1	2	8	2	1	1	1	1	?	4	8	8	8	8	8	8

If this ciphertext is decrypted and somehow made available to the adversary, the first half of the original message can be obtained via the procedure described in Section 2.1. Note that PGP will not return more data than the number of bytes described in the "Length" field of the header, while inserting random blocks effectively doubles the size of the message. Therefore, a straightforward implementation of the attack will require the adversary to obtain decryptions of *two* ciphertexts in order to recover the entire message (i.e., the adversary can obtain about half the plaintext with each decrypted ciphertext he receives). Alternately, the adversary can try to modify the "Length" field. The actual length of the packet is known, so by manipulating the bits of the "Length" field in the encrypted data the adversary can potentially set the value to the length of the modified ciphertext. This is possible, however it causes the next block of data to be garbled unpredictably (thereby preventing the adversary from learning the message corresponding to that block). Furthermore, this approach will be effective only if the lengths of the modified ciphertext and the original ciphertext lie in the same range; i.e., the lengths are described by the same number of bytes. Otherwise, the attack will fail because the adversary will have to insert bytes into the encrypted header of a PGP packet, which will result in unpredictable data.

Another minor complication is that PGP checks the value of the CTB after decryption. If the value of this byte is not valid, PGP will exit with an error message. Decryption will also fail if the "Mode" byte is not recognized. Thus, when constructing the chosen ciphertext message, the adversary must take care not to garble the first block of the message which contains the header for the Literal Data Packet. This is taken into account in the above diagrams.

Finally, PGP will read the number of bytes given in the "Name Length" field from the "Name" field; these bytes will not appear in the output plaintext (rather, they are used to derive a name for the file that will contain the plaintext). If the attacker inserts data in the encrypted packet before the end of the original filename, or if the filename does not end on a block boundary, the decrypted message will not properly align with the attacker's random data. This is so because the beginning of the decrypted chosen ciphertext will actually contain part of the filename field that is not normally output by PGP. This minor problem can be avoided by repeating the packet header blocks in the chosen ciphertext and then finding the proper alignment by shifting the decrypted message (i.e., by dropping off one byte at a time from the beginning of the message), thus repeatedly trying the attack until the "Length field" (whose value is known) is found. The alignment that allows determination of the "Length" field also allows the rest of the data to be determined, and no additional chosen ciphertext messages are required.

In summary, when the plaintext message is not compressed by PGP before encryption or when the plaintext is itself a compressed file which is not further compressed by PGP, a single-ciphertext or two-ciphertext attack can be used to determine the entire contents of the original message. A program (in Java) implementing the single-ciphertext and two-ciphertext versions of the attack is available at:

`http://www.cs.umd.edu/~jkatz/pgpAttack.tar.gz`

Because PGP and GnuPG inter-operate, the above attack also works against a message sent via GnuPG when compression is not used. The code has been tested with PGP versions 2.6.2 and 2.6.3 as well as GnuPG version 1.0.6.

4 Compressed Data

Both GnuPG and PGP compress data by default before encrypting it (unless the data is already compressed, as described above). Compression further complicates the attack because a compression algorithm is applied before encryption and after decryption. The difficulties arise from the fact that the programs use packet headers included in the encrypted data for further processing. Modification of these headers will usually cause the attack to fail because the program cannot process the garbled headers. Additionally, GnuPG uses a message digest check to protect the integrity of the data. While a minor modification of the attack succeeds against GnuPG , the digest check fails and causes the program to output a warning message that would likely cause the user to become suspicious and fail to return the decrypted chosen ciphertext message.

4.1 PGP

A PGP message with compressed data is formed as follows:

PGP COMPRESSED MESSAGE

CTB	LENGTH	PUBLIC KEY ENCRYPTED SESSION KEY DATA	CTB	LENGTH	RANDOM BYTES	LAST TWO RANDOM BYTES	CTB	ALG	COMPRESSED DATA
1	1	?	1	1	8	2	1	1	?

The compressed data is comprised of raw DEFLATE [4] blocks, the result of applying the DEFLATE algorithm to a PGP literal packet containing the plaintext.

The DEFLATE algorithm compresses data via a two-step process. First, it scans the data and finds "backward matches" or sections of the data that are redundant. These are represented by $\langle\text{length}, \text{distance}\rangle$ pairs that describe where in the data stream the redundancy occurs. Then, both the symbols in the data and the $\langle\text{length}, \text{distance}\rangle$ pairs are replaced by Huffman codes. Huffman coding essentially replaces each literal token with an encoded string, where literals that are more common in the actual data are given shorter codes. The "Huffman tree" is the mapping between literals and Huffman codes. This mapping is determined based on statistical analysis of the literals in the data, or is predefined by the algorithm.

The DEFLATE RFC [4] states the following:

> Any data compression method involves the reduction of redundancy in the data. Consequently, any corruption of the data is likely to have severe effects and be difficult to correct.

The chosen-ciphertext attack requires the insertion of random blocks into the encrypted data. Insertion or modification of any data in an encrypted stream will cause the plaintext of the next block to be random. Thus, in general, it is difficult to predict the effects of random changes in a compressed file because the compressed data is very dependent on the original data. Usually, random data in a compressed file will corrupt the internal state of the inflater during decompression and cause the algorithm to fail — the more random data inserted, the higher the probability that decompression will fail. If the decompression fails, a decryption of the chosen-ciphertext message is not produced and the attack cannot continue.

Based on some heuristic tests on about 400 text files, insertion of a single 64-bit block or modification of a block at random will cause a decompression error roughly 90% of the time. The larger the original message, the more likely it decompresses improperly after modification. Files that did decompress successfully were either severely truncated or were very similar to the original message.

It is claimed in [6] that if the attacker obtains the decompressed output resulting from the attack, he can re-compress it and continue the attack. This is not actually the case. Recall that the attacker needs the decryption of the chosen ciphertext in order to perform the attack. Normally, the payload of a PGP Session Key Encrypted Data Packet is an LDP which has been compressed and then encrypted:

$$C_1, C_2, \ldots, C_k = \mathcal{E}_K(\text{DEFLATE}(\text{LDP})).$$

Decryption and decompression proceed as follows:

$$P_1, P_2, \ldots, P_k = \text{INFLATE}(\mathcal{D}_K(C_1, C_2, \ldots, C_k)).$$

When $\mathcal{D}_K(C_1, C_2, \ldots, C_k)$ is in fact a valid output of the DEFLATE algorithm, we do indeed have

$$\text{DEFLATE}(\text{INFLATE}(\mathcal{D}_K(C_1, C_2, \ldots, C_k))) = \mathcal{D}_K(C_1, C_2, \ldots, C_k).$$

On the other hand, when the ciphertext is modified (as it is when the chosen ciphertext is constructed), $\mathcal{D}_K(C'_1, \ldots, C'_k)$ is no longer a valid output of the DEFLATE algorithm. The decompression process assumes that its input consists of Huffman codes that will translate into the appropriate literals for whatever Huffman tree is in use at that point of the decompression; in other words, INFLATE expects to run on input that is the valid output of the deflation algorithm. Thus (with overwhelming probability),

$$\text{DEFLATE}(\text{INFLATE}(\mathcal{D}_K(C'_1, \ldots, C'_k))) \neq \mathcal{D}_K(C'_1, \ldots, C'_k)$$

and the attack cannot proceed. If the decompression does produce a result that is not badly truncated or is very similar to the original message, it may be possible to reproduce at least part of $\mathcal{D}_K(C'_1, \ldots, C'_k)$ using the decompressed output and knowledge of where the random insertion was made. This requires careful analysis of the corrupted compressed message, which is difficult and in many cases impossible.

4.2 GnuPG

A variant of the chosen-ciphertext attack is partially successful against GnuPG, although the effectiveness of the attack is mitigated by the presence of an integrity check on the data. GnuPG uses slightly longer headers on compressed data, and these headers are predictable. This allows an attacker to change the header bytes of the data to those of a literal packet (by "flipping" bits in the encrypted data that correspond to bits in the known "underlying" value), although the data section is left alone. When the data is decrypted, the algorithm will not attempt to decompress the result because the compressed packet headers have been changed to literal packet headers. The decrypted chosen ciphertext will thus be the compressed version of the original message. Since compressed data bears no resemblance to the original data, the recipient will not be suspicious of the decrypted data and is likely to send it back.

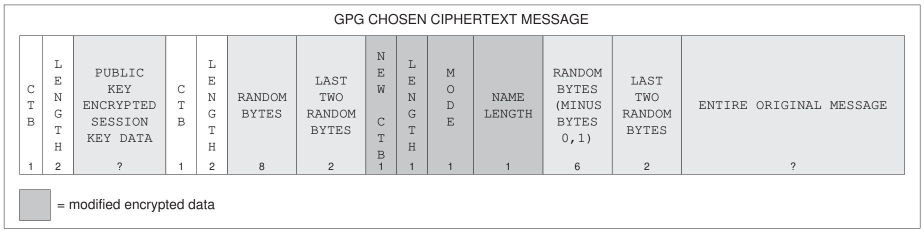
The key to this variant of the attack is that two extra bytes of the encrypted data are predictable under GnuPG's default compression algorithm, where the original message is as follows:

GPG COMPRESSED MESSAGE

CTB	LENGTH	PUBLIC KEY ENCRYPTED SESSION KEY DATA	CTB	LENGTH	RANDOM BYTES	LAST TWO RANDOM BYTES	CTB	ALG	METHOD	FLAGS	COMPRESSED DATA
1	2	?	1	2	8	2	1	1	1	1	?

The fact that the "METHOD" and "FLAGS" bytes are known in this case allows the attacker to flip bits in the underlying data to change the compressed packet header to a literal packet header, as in the diagrams detailing uncompressed data messages above.

In order to obtain the entire message, the attacker can set the filename length header to the blocksize of the algorithm and append the entire original segment of encrypted data to the (modified) first block as follows:



The result of decryption will be one block of random data, followed by the compressed packet. Upon receipt of the decrypted chosen ciphertext message, the attacker can strip off the garbage data and headers, and decompress the data to obtain the original message.

There is one flaw with the above approach. By default, GnuPG includes a message digest on the encrypted data. Since this check is likely to fail for the ciphertext constructed by the attacker, a warning message will be relayed to the user. This might make the user suspicious enough not to send the message back (thereby preventing the chosen-ciphertext attack). It should be noted, however, that the message digest is not used when GnuPG inter-operates with PGP, and this might allow the attack. When inter-operating with PGP, though, ZIP compression is used (instead of ZLIB); we were unable to implement the attack when ZIP compression is used.

5 OpenPGP Vulnerabilities

The OpenPGP specification is written as a base specification for security software inter-operability. Most implementations of the specification are written with inter-operability with PGP as a basic goal, and are built along similar lines. An application written "directly from specification" would potentially be vulnerable to a chosen ciphertext attack due to a difference in the specification and the actual operation of PGP. In defining constants allowed in the "Algorithm" field of a compressed data packet, the specification states the following [3]:

```
    ID   Algorithm
    --   ---------
    0    - Uncompressed
    1    - ZIP (RFC 1951)
    2    - ZLIB (RFC 1950)
    100 to 110  - Private/Experimental algorithm.

Implementations MUST implement uncompressed data. Implementations
SHOULD implement ZIP. Implementations MAY implement ZLIB.
```

The specification states that compliant implementations MUST implement uncompressed as one of their algorithms; that is, a compressed packet with an "Algorithm" field of 0 must be acceptable. In practice, neither GnuPG or PGP actually implement this; uncompressed data is never encapsulated in a compressed data packet. If this part of the specification were followed, however, an attack similar to the one described for GnuPG would be successful against that implementation — the attacker could change the encrypted compression algorithm header and obtain the compressed data as if it were plaintext. The attacker could then simply decompress that data and retrieve the original message. There is little chance that the user would recognize the data as "similar" to the actual message, because it would be in compressed format. The widely used programs (GnuPG and PGP) that claim to conform to the specification actually do not in this instance, and therefore fortuitously escape vulnerability to the attack.

In general, the OpenPGP standard presents compression as optional. An implementation that did not provide compression would be vulnerable to the non-compressed attack as described.

It is also important to note that the OpenPGP standard does not explicitly require an integrity check on the contents of an encrypted message.[5] An implementation of GnuPG which did not include the integrity check would be vulnerable to chosen-ciphertext attack. GnuPG includes this integrity check by defining their own packet type which is not included in the OpenPGP standard. When this packet type is encountered by the program, it actually uses a slightly different chaining mode (the re-sync step is omitted). This prevents an attacker from changing the headers on a packet with an integrity check to make it look like a non-integrity checked packet: such a packet will not decrypt properly due to the difference in chaining modes used. This is another example of an extension to the OpenPGP standard that allows the program to avoid vulnerability to chosen-ciphertext attacks in practice.

6 Recommendations

If compression is not used, or if compressed files are sent, the chosen-ciphertext attack described here succeeds against both GnuPG and PGP. GnuPG is also vulnerable if the user does not view the warning message that the encrypted data fails the message integrity check. In "batch mode" operation this warning would probably go unnoticed by the user; since in this case the decrypted file is still produced, the attack would succeed. Additionally, some of the front-end programs that use GnuPG do not propagate this warning to the user. In this case, the attack is definitely feasible.

Users of GnuPG and PGP should be aware that compression should not be turned off. Compression is turned on by default, but a user sending a compressed file will still be at risk from a chosen-ciphertext attack.

The OpenPGP standard, as written, is vulnerable to chosen ciphertext attack due to the following:

[5] But see footnote 2.

1. No explicit requirement of a message integrity check.
2. Optional implementation of compression.
3. Requiring acceptance of "uncompressed" as a valid form of compression.

The first problem is basically a recognized one that has been solved in practice by those implementing the algorithm. On the other hand, precisely because the problem is universally recognized by those "in the know", it is time for the RFC to reflect this. Requiring "uncompressed" to be recognized as a valid form of compression is a minor, but clear, problem with the standard itself. Compression algorithms already allow for inclusion of non-compressed data and the standard should not try to deal with this issue because it introduces a flaw. Luckily the widely used programs that (generally) conform to the standard ignore this requirement; the standard should still be fixed so as not to cause problems in new implementations.

Developers of front-end software for GnuPG need to propagate integrity violation warnings to the users. This is important not only for protection against chosen ciphertext attacks — integrity protection is useless if the user is not warned when it has been violated!

Acknowledgments

Thanks to Jon Callas and David Shaw for their extensive comments and helpful suggestions.

References

1. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. Crypto '98.
2. J. Callas and D. Shaw. Personal communication, July 2002.
3. J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. "OpenPGP Message Format," RFC 2440, Nov. 1998.
4. L.P. Deutsch. "DEFLATE Compressed Data Format Specification version 1.3," RFC 1951, May 1996.
5. S. Garfinkel. *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995.
6. J. Katz and B. Schneier. A Chosen Ciphertext Attack against Several E-Mail Encryption Protocols. 9th USENIX Security Symposium, 2000.
7. M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. STOC '90.
8. P. Zimmerman. *The Official PGP User's Guide*, MIT Press, 1995.