IACR ePrint Archive 2004/078.

Analysis of the WinZip encryption method

Tadayoshi Kohno*

March 29, 2004

Abstract

WinZip is a popular compression utility for Microsoft Windows computers, the latest version of which is advertised as having "[e]asy-to-use AES encryption to protect your sensitive data." We exhibit several attacks against WinZip's encryption method, dubbed AE-2. We then discuss secure alternatives. Since at a high level the underlying WinZip encryption method appears secure (the core is exactly Encrypt-then-MAC using AES-CTR and HMAC-SHA1), and since one of our attacks was made possible because of the way that WinZip Computing, Inc. decided to fix a different security problem with its previous encryption method AE-1, our attacks further underscore the subtlety of designing cryptographically secure software. We also identify problems with PKZIP's new encryption method, dubbed EFS; since the problems with PKZIP are either more classic, and therefore less informative though not necessarily less serious, or shared with WinZip, we focus our discussion on the WinZip encryption method.

Keywords: WinZip, PKZIP, Zip, attacks, security fixes.

*Dept. of Computer Science and Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: tkohno@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/tkohno. Supported by a National Defense Science and Engineering Graduate Fellowship.

1 Introduction

WinZip [24] is a popular compression utility for Microsoft Windows computers, the latest version of which is advertised as having "[e]asy-to-use AES encryption to protect your sensitive data" [24]. Because of WinZip's already established large user base, and because of its advertised encryption feature, we anticipate that many current and future users will choose to exercise this encryption option in an attempt to cryptographically protect their data.

Unfortunately, there are a number of subtle problems with WinZip's latest encryption scheme, dubbed AE-2 [23] and shipped with WinZip 9.0. We exhibit several attacks in this note, and then propose ways of fixing the protocol. We believe that our proposed fixes are relatively non-intrusive and will be easy to incorporate into WinZip and other WinZip-compatible applications.

WinZip. We shall write "WinZip" when we mean "WinZip 9.0" or any other recent version of WinZip or a WinZip-compatible tool that uses the AE-2 encryption scheme [23].[1] When archiving a file, if the length of the file is above some threshold, WinZip first compresses the file using some standard compression method such as DEFLATE [7]. WinZip then invokes the AE-2 encryption method on the output of the previous stage. Specifically, it derives AES [6] and HMAC-SHA1 [16] keys from the user's passphrase and then encrypts the output of the compression stage with AES in counter (CTR) mode (AES-CTR) and MACs the resulting ciphertext with HMAC-SHA1. The underlying AES-CTR-then-HMAC-SHA1 core is a provably secure authenticated encryption scheme per results by Bellare and Namprempre [1] and Krawczyk [16] and standard assumptions on AES-CTR and HMAC-SHA1.

A collection of issues. All our attacks exercise different problems with the way that WinZip attempts to protect users' files. Furthermore, each of the attacks accomplish something slightly different and under different attack models, which means that different adversaries may prefer different attacks. Since no single "best" attack exists, since in order to eventually fix the protocol we must first understand the (orthogonal) security issues with the current design, and since we believe that each of the issues we uncover is informative, we discuss each of the main problems we found, and their corresponding attacks, in turn. But first we put our attacks in perspective:

Perspective. WinZip's new encryption method is still significantly better than the traditional Zip encryption method [9], which Biham and Kocher [5] and Stay [21] have thoroughly cryptanalyzed. Indeed, it is worth noting that although our attacks can reveal a significant amount of information about WinZip-encrypted files to an adversary under some realistic scenarios, including in some cases the entire contents of the original files, none of our attacks "totally break" WinZip in the sense that they *do not* enable an adversary to recover a passphrase or an underlying AES key simply from a single WinZip archive or a set of archives. Rather, besides information leakage through an encrypted file's header, our more devastating attacks require an adversary to be able to modify portions of an encrypted WinZip archive between when a user creates it and when the user, or another user sharing the same passphrase, later attempts to decrypt it. This means that, unless some brilliant discovery is made against AES, no one will be able to write a program that will be able to reconstruct the entire contents of an encrypted WinZip file simply from the archive itself, unless of course the user chose a guessable passphrase like "Alice" or "secret." This should be very reassuring to many WinZip users. Nevertheless, we believe that security protocols should be secure under all reasonable attack scenarios, such as the ones we use in this paper, which means that we consider our attacks significant and worth protecting against.

[1] According to the documentation packaged with WinZip 9.0, "Because the technical specification for WinZip's AES format extension is available on the WinZip web site, we anticipate that other Zip file utilities will add support for this format extension."

We have also identified a number of issues with another major Zip utility's new encryption method, PKZIP's EFS [18], which is different than WinZip's AE-2.[2] Although we discuss our findings with EFS briefly, we focus on WinZip in this paper because we believe that our specific attacks against WinZip are more instructive. For example, since PKZIP does not include a MAC in their construction at all, the presence of integrity attacks against EFS-encrypted files is not surprising. (Note that an integrity attack can lead to a chosen-ciphertext privacy attack.) On the other hand, WinZip's AE-2 method *does* combine a MAC with an encryption scheme using the well-studied Encrypt-then-MAC paradigm [1, 16], and therefore the initial reaction of an outside reviewer might be that integrity attacks against AE-2-encrypted archives are impossible. We show, however, that because of a subtle issue with how WinZip's Encrypt-then-MAC core is used, integrity attacks against AE-2 encrypted archives are in fact possible. Furthermore, it is worth noting that a number of the issues that we raise with respect to WinZip are also directly applicable to PKZIP.

Information leakage from encrypted files' headers. According to the WinZip documentation, there is a known problem with the WinZip encryption architecture in that the header of an encrypted file appears in the WinZip archive in cleartext. Contained in this header is the encrypted file's original file name, the file's last modification date and time, the length of the original plaintext file, and the length of the resulting ciphertext data, the latter also being the length of the compressed plaintext data plus some known constant. Although we understand that WinZip Computing, Inc. may have had engineering, usability, and backward-compatibility reasons for leaving these fields unencrypted, the risks associated to leaving these fields unencrypted should not be discounted. For example, if the name of a compressed and encrypted file in the `PinkSlips.zip` archive is `PinkSlip-Bob.doc`, encrypting the files in the archive will not prevent Bob from learning that he may soon be laid off. Additionally, a recent result from Kelsey [14] shows that an adversary knowing both the length of the uncompressed plaintext file and the length of the compression output will be able to learn some information about the original plaintext. For example, from the compression ratio an adversary might learn the language in which the original file was written [3]. Of course, the mere name, date, and size of a `.zip` archive may reveal information to an adversary, so the goal here should not be to prevent all information leakage, but to reduce the amount of information leakage whenever possible.

Information leakage through an encrypted file's unencrypted header is not unique to WinZip's new AE-2 encryption method. Indeed, the traditional Zip encryption method, as well as PKZIP's new EFS encryption method, also leak the same information that WinZip leaks. In the case of EFS, this is most likely for the same engineering, usability, and backward-compatibility reasons. In addition to the above-mentioned fields, however, the traditional Zip encryption method and PKZIP's EFS method also leak the 32-bit CRC of the original plaintext file, which reveals even more information about the original file to an adversary looking at the resulting `.zip` archive. An adversary could, for example, use the CRC to verify a guess of an encrypted file's contents.

Interactions between compression and the AE-2 encryption method. One of our chosen-ciphertext attacks exploits a novel interaction between WinZip's compression algorithm and the AE-2 encryption method. In particular, although the underlying AES-CTR-then-HMAC-SHA1 core of AE-2 provably protects both the privacy and the integrity of encapsulated data, cf. [1] and [16], an attacker can exploit the fact that the header fields indicating the chosen compression method and the length of the original file are *not* authenticated by HMAC-SHA1 as part of AE-2.

An example situation in which an adversary could exploit this flaw is the following: Two parties, Alice and Bob, wish to use WinZip to protect the privacy and integrity of some corporate data. To

[2] For a discussion of the splintering of the Zip encryption formats, see [20].

do this, they first agree upon a shared secret passphrase. Suppose Alice uses WinZip to compress and encrypt some file `F`, using their agreed upon passphrase to key the encryption, and let `F.zip` denote the resulting archive. Now suppose Alice sends `F.zip` to Bob, perhaps through email or by putting it on some corporate file server or an anonymous ftp server.

We argue that the type of security that Alice and Bob would expect in this situation is very similar to the notion of authenticated encryption introduced by Bellare and Rogaway [2] and Bellare and Namprempre [1]. Unfortunately, an adversary, Mallory, could break the security of WinZip under this model. For example, assume that Mallory has the ability to change the contents of `F.zip`, replacing it with a modified version, `F-prime.zip`, that has a different value in the header field indicating the chosen compression method and an appropriately revised value for the plaintext file length. When Bob tries to decrypt and uncompress `F-prime.zip`, he will use the incorrect decompression method and will get back what looks like completely unintelligible garbage `G`. Now suppose that Mallory can obtain `G` in some way. For example, suppose Bob sends the frustrated note "The file you sent was garbage!" to Alice. If Mallory intercepts that note, he might reply to Bob, while pretending to be Alice, "I think I've had this problem before; could you send the garbage that came out so that I can figure out what happened; it's just garbage, there's no reason not to include it in an email." Mallory, after obtaining `G`, can reconstruct Alice's original data `F`.

We believe that the above attack scenario is in some cases quite realistic. The following quote, appearing on two of WinZip Computing, Inc.'s websites [22, 25], shows that WinZip Computing, Inc. expects some users to mail encrypted Zip files to other users, thus satisfying one possible precondition of our attack:

> "Note: Recipients to whom you send AES-encrypted Zip files must have a compatible Zip file utility in order to decrypt the files [22, 25]."

It also appears that employees of at least one large corporation basically used Alice's and Bob's approach for sharing confidential files: Diebold Election Systems employees transported important election-related files, compressed and encrypted into Zip archives, via an anonymous ftp site [11]. Given Jones' [11] discussion of Diebold's procedures, we would be surprised if an adversary able to modify `F.zip` could not also get access to the decrypted, garbage-looking file `G`. Additionally, we remark that security products should ideally remain secure even in the face of potential misuses by non-security conscious users, which supports our claim that the attack we describe is realistic and should be protected against. Finally, we remark that our attack scenario is basically the same scenario that Katz and Schneier [13] and Jallad, Katz, and Schneier [10] used when attacking email encryption programs and PGP.

It is worth noting that the attack discussed in this section can also be mounted against a single user, for example if the attacked user attempts to discuss her problems with `help@winzip.com` and if Mallory can intercept those communications.

The attack discussed in this heading was made possible because of the way that WinZip Computing, Inc. decided to fix a different problem with its previous encryption method, AE-1. Details in the body of this paper.

On the names of files and their interpretations. There are a number of systems that associate software applications with file names; for example, a Microsoft Windows machine will by default open `.doc` files with Microsoft Word and `.ppt` files with Microsoft Power Point. Unfortunately, WinZip's AE-2 encryption method, as well as the traditional Zip encryption method and PKZIP's new EFS method, do not authenticate the file name field of an encrypted file's header, meaning that Mallory could modify the names of the encrypted files in an archive without triggering any detection mechanism within the extraction utility. This is problematic since, on a system like

Microsoft Windows, it is important for an extracted file to have the same extension as the original file. Otherwise, when Bob tries to open that file, he will accidentally use the wrong application, most likely get an error message, and thereby possibly allow Mallory to mount an attack similar to the one described in the previous heading. Note that the issue described here is orthogonal to the issue of leaving an encrypted file's file name unencrypted; specifically, the issue is not that the file name is stored in cleartext, but that the file name is not authenticated.

As a related attack, an adversary might benefit from renaming the files in an archive while keeping the same file name extensions. For example, Mallory might benefit from changing the headers of the files in the `Salaries.zip` archive so that, upon extraction, the file previously named `CEO-Salary.dat` actually extracts to `Mallory-Salary.dat`.

The lesson here is that a file encryption utility must not only protect the integrity of the encapsulated data, but must also protect the integrity of all of the "metadata," like the file name or file name extension, necessary for the surrounding system to correctly interpret that data.

Interactions with AE-1 and a chosen-protocol attack. According to the WinZip AE-2 specification [23], the AE-2 encryption method fixes a security problem with an earlier AE-1 encryption method. Further, according to [23], software implementing the AE-2 encryption method must be able to decrypt files encrypted with AE-1. While AE-2 does protect against a specific attack against AE-1, there is unfortunately a chosen-protocol attack against WinZip that exploits the fact that an adversary can force WinZip to use the AE-1 decryption method on an AE-2-encrypted file.

The attack works in the same setting as the previous attacks. In this attack, Mallory intercepts `F.zip`, makes a guess of `F`, and creates a replacement `F-prime.zip` based off his guess. If Bob can successfully decrypt `F-prime.zip`, e.g., if Bob doesn't complain to Alice that the file failed to decrypt, then Mallory learns with high probability whether his guess for `F` was correct. To compare this attack with the previous attack, note that Mallory only needs to learn whether `F-prime.zip` decrypted successfully. On the other hand, Mallory only learns whether his guess for `F` was correct. Still, this may constitute a serious attack if Mallory knows that `F` is from a small set of possible values, perhaps because of pre-existing knowledge of the message space or additional information gleaned from the compression ratio, and wants to know which value it is. (Actually, in some situations Mallory may learn more than just whether his guess was correct; details in the body of this paper.)

Key collisions and repeated keystream. To encrypt a file, WinZip first takes the user's passphrase and derives cryptographic keys for AES and for HMAC-SHA1. The key derivation process is randomized; one of the reasons for this randomization is so that two different files encrypted with the same passphrase will use different AES and HMAC-SHA1 keys. Unfortunately, because not enough randomness is used in the key derivation process, we expect key collisions after encrypting only 2^{32} files when using AES with 128-bit keys (2^{48} files for 192-bit AES and 2^{64} files for 256-bit AES). Furthermore, the AE-2 specification says that the initial CTR mode counter is always zero.[3] Thus, we can expect keystream reuse after encrypting only around 2^{32} files, which is much less than the 2^{64} files we would expect if we chose a different random key for each file. Additionally, assuming that the encrypted files are all of realistic size, then this is also less than the number of files we would expect if we used AES in CTR mode with just a single key but a randomly selected initial counter for each file.

It is worth noting that because WinZip encrypts each file in an archive independently, all 2^{32}

[3] Previously we said that the underlying Encrypt-then-MAC core of AE-2 is a provably secure authenticated encryption scheme per Bellare and Namprempre [1] and Krawczyk [16]. Because the initial CTR mode counter is always zero, we were assuming that each key is used to encrypt at most one message, which is typically the case assuming that less than 2^{32} files are encrypted per passphrase.

files need not be put into separate archives; we expect keystream reuse even if all 2^{32} files are distributed amongst only a small set of WinZip archives.

The problems with keystream reuse are well known: Once Alice reuses keystream, Mallory will be able to learn information about the compressed and encrypted plaintext. In the worst-case scenario, if Mallory knew the entire content of the larger, after compression, of two files encrypted with the same keystream, then Mallory would immediately know the entire contents of the other file.

Other ways of attacking WinZip. We remark that there are other ways in which an adversary might attack WinZip, PKZIP, or any other compression utility. For example, as noted in the WinZip documentation, an adversary might try to capture a user's passphrase by installing a keyboard logger on the user's computer or might try to resurrect a plaintext file from memory. We also observe what we believe to be a novel integrity attack against self-extracting password-protected executables: An adversary wanting to replace the data encapsulated by a password-protected self-extracting executable could write a new executable, with a similar user interface to the real self-extracting executable, that ignores the user-entered passphrase and simply creates a data file of the adversary's choice. However, attacks such as these are unrelated to the AE-2 encryption method, and since our focus is on the AE-2 encryption method we do not consider these attacks further.

Secure alternatives. Motivated by the issues and attacks discussed above, we discuss how to fix the WinZip encryption algorithm while simultaneously minimizing the changes to the AE-2 specification. Full proofs and definitions, written so as to be generically applicable to file compression utilities, to appear in the full version of this paper. For our definitions, we use a notion related to authenticated encryption [1, 2, 19], the main difference being that we take into consideration the fact that the output of the compression stage will typically have a different length than the plaintext input to the compression stage.

Issues with PKZIP's new encryption method. As remarked earlier, we also identified a number of issues with PKZIP's new encryption method, EFS [18], but chose not to focus on those issues in this paper. First, we observe that the EFS specification [18] is much less complete than the AE-2 specification [23]; in fact, it is impossible to know exactly how EFS works from the specification alone. Nevertheless, we can reach a number of security-related conclusions. We have already observed that, like WinZip's AE-2 and the traditional Zip encryption method, PKZIP's EFS does not encrypt the header fields of encrypted files. Unlike WinZip's AE-2, however, but like the traditional Zip encryption method, the EFS format also leaks the 32-bit CRC of the original plaintext file. We have also noted that an adversary could undetectably modify the file name of an EFS-encrypted file and thereby mount a chosen-ciphertext attack that leverages the fact that many systems will, for example, by default open `.doc` files and `.ppt` files with different applications.

We now observe what we believe to be a more fundamental problem with the EFS encryption method: It does not employ a message authentication code at all. The most likely reason for this is either that the makers of PKZIP did not realize the threat of chosen-ciphertext or integrity attacks, or that the makers of PKZIP did not realize that classic encryption techniques, like a block cipher in CBC mode or a stream cipher, are not designed to protect the integrity of encapsulated data. With respect to the encryption methods themselves, we note that in addition to encryption methods like 3DES and AES in CBC mode, the EFS specification includes 40-, 64-, and 128-bit RC4 as possible encryption methods; also included are 40-, 64-, and 128-bit RC2 and single DES in CBC mode. The allowable short key lengths are obviously of concern, but so is the fact that there are distinguishing attacks against RC4 requiring on the order of a gigabyte of output [8]. Furthermore, the EFS specification does not describe exactly how RC4 is to be used, and it is therefore not clear whether recent results, like Mironov's [17], are taken into consideration. We also note that to encrypt the files in an archive the EFS method first generates a random value RD, which is then used to derive a different encryption key for each file in an archive. Unfortunately, the EFS specification does not describe how long the RD field must be. This is a concern since if the length of RD is less than the key length of the underlying cipher, which we imagine might be the case, perhaps when the block cipher is three-key 3DES, then an adversary attacking the privacy of the system could simply mount an exhaustive search for RD, rather than for the underlying block cipher key. Of course, an exhaustive search for the user's passphrase might still be more efficient.

Additional related works. Biham [4] introduced the notion of key collision attacks in the context of DES, noting that we expect one key collision after encrypting about 2^{28} messages using randomly selected 56-bit DES keys. Our key collision attack accomplishes the same goal, seeing data encrypted with the same key, but is more efficient than a normal key collision attack because of the way that WinZip derives AES keys from passphrases. Kelsey, Schneier, and Wagner [15] introduced the concept of a chosen-protocol attack.

2 The WinZip compression and encryption method

WinZip's underlying compression architecture follows the Info-ZIP specification [9]. The AES-based AE-2 encryption algorithm is described on WinZip's website [23]. The difference between the AE-2 encryption algorithm and the AE-1 encryption algorithm is slight and will be mentioned below.

A Zip archive can contain multiple files. At a high-level, each file in an archive is compressed and encrypted independently, typically using the same passphrase for the encryption of each file, and the resulting compressed and encrypted files are concatenated to form the resulting multi-file archive. Additional information, including copies of parts of the headers from the individual encrypted files, is added to the end of the resulting concatenated file. Since it is not critical to understand the structure of the WinZip archive trailer, we focus on how an individual file is compressed and encrypted. We stress, however, that in actually mounting some of our attacks that require modifying portions of a file's header, the adversary may also have to modify the identical header information mirrored at the end of the archive.

When referring to fields of an encrypted file's header, byte strings will be written like $\texttt{504b0304}_{\rm bs}$, meaning that the first byte is $\texttt{50}_{\rm bs} = 80$, the second byte is $\texttt{4b}_{\rm bs} = 75$, and so on. Integers, such as lengths, that are stored in multi-byte fields are encoded in little endian format.

Per-file header information. According to the Info-ZIP specification [9], barring certain extensions, all compressed files have the following structure (the fields important to our work are highlighted): file header signature (4 bytes, always $\texttt{504b0304}_{\rm bs}$), version needed to extract (2 bytes), general purpose bit flag (2 bytes), *compression method* (2 bytes), *last modification time* (2 bytes), *last modification date* (2 bytes), *32-bit CRC* (4 bytes), compressed size (4 bytes), *uncompressed size* (4 bytes), file name length (2 bytes), extra field length (2 bytes), *file name* (variable size), and *extra field* (variable size). Following the header fields is the *file data* itself. We describe how the file data looks when it is compressed and encrypted shortly, but first we describe the contents of the extra field when the AE-2 encryption method is used.

AE-2 settings and the AE-2 extra data field. When the AE-2 WinZip encryption algorithm is turned on, the four bytes reserved for the 32-bit CRC are set to zero, bit 0 of the general purpose flag is set to 1, and the two bytes reserved for the compression algorithm indicator are set to $\texttt{6300}_{\rm bs}$. The extra data field of the above header will consist of the following 11 bytes (again, important fields highlighted): extra field header id (2 bytes, always $\texttt{0199}_{\rm bs}$), data size (2 bytes, $\texttt{0700}_{\rm bs}$ for AE-

2 since there are seven remaining bytes in the 11-byte extra data field), *version number* (2 bytes, always $\texttt{0200}_{\text{bs}}$ for AE-2), 2-character vendor ID (2 bytes, always $\texttt{4145}_{\text{bs}}$ for AE-2), value indicating AES encryption strength (1 byte), and *the actual compression method used to compress the file* (2 bytes). The encryption strength field will be $\texttt{01}_{\text{bs}}$ (resp., $\texttt{02}_{\text{bs}}$ or $\texttt{03}_{\text{bs}}$) if the file is encrypted with AES using a 128-bit (resp., 192-bit or 256-bit) key. Example values for the actual compression method are $\texttt{0800}_{\text{bs}}$ if the file is DEFLATEd [7] and $\texttt{0000}_{\text{bs}}$ if no compression is used.

File data. Following the formal extra data field described above, we have the following additional AE-2-specific fields, formally encoded as part of the file data field. These additional fields are: salt (variable length), password verification value (2 bytes), encrypted file data (variable length), and the authentication code (10 bytes). The salt is 8 bytes (resp., 12 bytes or 16 bytes) long if the AES key is 128 bits (resp., 192 bits or 256 bits) long.

Encryption and authentication. Before applying the AE-2 encryption method, assuming that the length of the plaintext file is above a certain threshold and that the user has not chosen otherwise, the plaintext file is compressed. Then an AES encryption key, an HMAC-SHA1 key, and a password verification value are derived from the user's passphrase and a salt using the PBKDF2-HMAC-SHA1 algorithm [12]. The length of the salt depends on the chosen length of the AES key and is specified above. The specification [23] states that the salt should not repeat, and since this must be true across different invocations of the compression tool, suggests making the salt a random value.

The derived AES key is used to encrypt the compressed data using AES in CTR mode with zero as the initial counter. The compressed plaintext data is not padded before encryption. After encryption, the encrypted data is MACed using HMAC-SHA1 and the derived MAC key, and 80 bits of the HMAC-SHA1 output are used as the authentication code.

AE-1. The only differences between the AE-2 method and the earlier AE-1 method is that in AE-1 the version number in the extra data field is $\texttt{0100}_{\text{bs}}$ and the 32-bit CRC field in the file's header is not all zero but actually contains the CRC of the original unencrypted data, which must be checked upon extraction. The motivation for zeroing out the CRC field in AE-2 is because the CRC of the plaintext will leak information about the plaintext.

3 Header-based information leakage

As discussed in Section 1, the header leaks information in several ways. The names of the encrypted files are stored in cleartext. This is a concern since many users will probably not expect this behavior. The files' last modification dates and times are also stored unencrypted. Additionally, the length of plaintext files are stored in the files' headers unencrypted. This is a concern since, as Kelsey showed in [14], an adversary can learn information about the plaintext simply given the lengths of both the original and the compressed data. As Kelsey notes, information leakage via the compression ratio of files becomes particularly effective if Mallory has pre-existing partial knowledge of the plaintext or if Mallory can see the compression ratio of multiple related files, e.g., different versions of the same file over time. The WinZip documentation notes that these pieces of information are included unencrypted in the header. However, the risks associated with leaving these fields unencrypted is not considered.

It is worth noting that theses fields are left unencrypted most likely for engineering, usability, and backward-compatibility reasons. As such, it is not surprising that other Zip implementations, such as PKZIP, also leave these header fields unencrypted. Note, however, that whereas WinZip's AE-2 mandates zeroing out the field of an encrypted file's header corresponding to the CRC of the original plaintext file, PKZIP's new EFS encryption method does not.

4 Exploiting the interaction between compression and encryption

Recall the setup described in Section 1. The critical observation for this attack is that despite the fact that the underlying encryption core is a provably secure Encrypt-then-MAC authenticated encryption scheme, cf. [1, 16], the compression type indicator and the length of the original file, which are stored in the encrypted file's header, are *not* authenticated. This means that an adversary can change the compression type and uncompressed file length without voiding the HMAC-SHA1 authentication tag attached to the file. Consequently, assuming that the new uncompressed file length field is correct or that the extraction tool does not check that field, when Bob attempts to decrypt and decompress the modified file `F-prime.zip`, the MAC verification will succeed and the user will not see any error. But because the adversary changed the compression type, the file will be decompressed using the wrong algorithm and the resulting file `G` will look like garbage.

In the most natural case, one would probably change the compression type from `0800`$_{\text{bs}}$, which appears to be WinZip's default and which corresponds the DEFLATE algorithm [7], to `0000`$_{\text{bs}}$, which corresponds to no compression. This is very easy to do and very efficient — simply parse the file and change the compression type and the uncompressed file length fields in the file's header and in the trailer at the end of the WinZip archive. After Bob runs the WinZip extraction tool on `F-prime.zip` to get garbage `G`, which he accidentally passes to Mallory in some way, Mallory can run the DEFLATE algorithm on `G` himself and recover the original data.

We implemented this attack against WinZip 9.0. To create `F-prime.zip` from `F.zip`, rather than parse `F.zip` and switch the compression type from `0800`$_{\text{bs}}$ to `0000`$_{\text{bs}}$, we found that the Unix `tcsh` command line

```
cat F.zip | sed ’s/\(\x02\x00\x41\x45\x01\)\x08\x00/\1\x00\x00/g’ \
          > F-prime.zip
```

was sufficient in all of the cases that we tried, showing that the attack is indeed very easy to mount. We would only expect the above command line to not work as desired if the 7-byte string `02004145010800`$_{\text{bs}}$ appears in `F.tar` in a place not corresponding to the extra data field of a file's header or in the mirror of the extra data field at the end of the WinZip archive. Since the WinZip 9.0 extraction tool did not seem to verify the length of the extracted file, we did not need to modify the field of a file's header corresponding to the length of the uncompressed file.

Recall that in AE-1 the CRC field of an encrypted file's header contains the CRC of the original plaintext file but that the field is all zero in AE-2. When trying to mount the above attack against AE-1, since the extraction utility will also verify the CRC of the plaintext, which will typically fail because the plaintext is now different, the resulting garbage-looking file `G` will not be saved and the attack will not immediately go through. While it is true that if Bob is crafty he may be able to view `G` among the temporary files created by WinZip during the extraction process and before the CRC failure is noted, send `G` to Alice, and thereby leak `G` to Mallory, it would probably be unrealistic for Mallory to assume that Bob will find `G` among WinZip's temporary files. This discussion highlights the subtlety of cryptographic design since the vulnerability presented in this section was accidentally introduced when the authors of the specification tried to fix a different problem with AE-1.

5 Exploiting the association of applications to file names

To complement the attack in Section 4, we note that on many systems, including Microsoft Windows machines, software applications are automatically attached to files based off the files' extensions; e.g., Microsoft Windows will by default open `.doc` files with Microsoft Word. Since the file name field of an encrypted file's header is unauthenticated, an adversary could modify that field without voiding the MAC included at the end of the encrypted file's record. Once Mallory does this, he can effectively mount a variant of the attack in Section 4. This shows that a file encryption utility must not only protect the integrity of the encapsulated data itself, but also the metadata, like the file name extension, necessary for the surrounding system to correctly interpret that data. Note that the problem described here is also shared with PKZIP's new EFS encryption method, as well as the traditional Zip encryption method.

We also observe that an adversary could benefit from changing the names of the encrypted files in an archive while still maintaining the files' original extensions. E.g., if Alice's salary is currently higher than Mallory's, Mallory could swap the names of the files `Alice-Salary.dat` and `Mallory-Salary.dat` in an encrypted archive `Salaries.zip` without triggering any detection mechanism within the WinZip extraction utility.

6 Exploiting the interaction between AE-2 and AE-1

The motivation for the change from AE-1 to AE-2 is that in AE-1 the CRC of the plaintext will leak information about the plaintext. While this is true, one can exploit the interaction between AE-1 and AE-2 in the following chosen-ciphertext attack that reveals information about an AE-2-encrypted file's CRC to an adversary, though not as conveniently as with AE-1. Note that according to the AE-2 specification [23], Zip tools that understand AE-2 must be able to decrypt files encrypted with AE-1 and must verify the CRC upon extraction.

Recall again the setup described in Section 1. Assume Alice sends the encrypted file `F.zip` to Bob, but assume that Mallory can modify the file in transit and can learn whether Bob can successfully extract the file he receives using the passphrase he shares with Alice. Now suppose that Mallory has a guess for what the original contents of `F` are, but is not completely sure and wants to verify his guess `Guess`. He can do this as follows: Compute the 32-bit CRC of `Guess` and then modify `F.zip` such that the version number in the extra data field is `0100`$_{\text{bs}}$ and the CRC field in the header has the CRC of `Guess`. Let `F-prime.zip` denote the Mallory-doctored file. If Mallory's guess is correct, then Bob will be able to extract `F` from `F-prime.zip` without any error. Otherwise, Bob will with high probability see an error dialog box like the following, which is the error we received when mounting this attack with an incorrect guess and then trying to extract `F-prime.zip` using WinZip 9.0:

```
Data error encountered in file
          C:\F
Possibly recoverable, contact help@winzip.com and mention error code 56.
```

By observing Bob's reaction, Mallory will learn whether his guess was correct.

Although not necessarily the case with all Zip tools but in the case of WinZip, after dismissing the initial error dialog box Bob will have the option of viewing a more detailed error log. If Bob chooses to see this error log, he will see a line like the following:

```
bad CRC 1845405d  (should be 1945405d)
```

If Bob decides to copy and paste this detailed error message in an email to Alice or `help@winzip.com`, and if Mallory sees this email, then Mallory will learn the CRC of plaintext file, and thereby learn additional information about the plaintext.

If we look more closely at how WinZip behaves when it attempts to extract a modified file with an incorrect CRC guess, it appears that the file is first extracted, the CRC is checked, the user is told that the CRC check failed, and then the extracted file is deleted. This means that if Bob is crafty he will be able to access the unencrypted file between when it is extracted and when it is automatically deleted after the CRC check fails. Even if Bob does this, which we expect to be unlikely, he may not be confident in the correct extraction of the file and may convey this lack of confidence to Alice.

We again stress that for WinZip's previous encryption method AE-1, as well as for the traditional Zip encryption method and for PKZIP's new encryption method EFS, the CRC of the plaintext data is stored unencrypted in the header of an encrypted file, meaning that the CRC is readily available to anyone looking at an AE-1- or EFS-encrypted archive. I.e., in the case of information leakage through the CRC, despite the attack we present here, the AE-2 encryption method is an improvement over the alternative Zip encryption methods.

7 Repeating keystream and on the password-based key derivation

When AE-2 is used with a 128-bit AES key, then one can expect CTR mode keystream reuse after encrypting approximately 2^{32} files, which is much less than one would normally expect given 128-bit AES keys and 128-bit blocks. (When using 192-bit AES keys with AE-2, we expect keystream reuse after encrypting 2^{48} files; when using 256-bit AES keys, we expect collisions after encrypting 2^{64} files). The security problems with reusing keystream are well-known.

This problem arises for two reasons. First, the salt used when deriving the AES and HMAC-SHA1 keys from the passphrase is only 64 bits (resp., 96 bits and 128 bits) long when the desired AES key length is 128 bits (resp., 192 bits and 256 bits). Second, AES-CTR is specified to always use zero as the initial block counter. The former means that, with 128-bit keys, after encrypting 2^{32} files we expect there to be one AES key that we used twice. The latter means that when we use the same AES key twice, we will use the same keystream both times.

Changing topics slightly, one of the reasons for using PBKDF2 [12] and a salt when deriving AES and HMAC-SHA1 keys from passphrases is to impede dictionary attacks. Specifically, an exhaustive search through the most common passphrases will be very slow because of the computational requirements for PBDKF2, and a dictionary of HMAC-SHA1 keys, corresponding to the most common passphrases and all possible salt values, will be extremely large because of the number of possible salt values. But we make the observation that since a different salt is used to encrypt each file, an adversary may not need to use *all* possible salt values when populating an HMAC-SHA1 key dictionary. In particular, Mallory would only need to populate the dictionary using enough different salt values to ensure, with high probability, that one of the salt values that a user uses when encrypting her files will collide with one of the salt values that Mallory used when creating his dictionary. For example, if the salt is 8 bytes long and if each user is expect to encrypt on the order of 2^{32} files, then Mallory would only need to use 2^{32} different salt values when creating his HMAC-SHA1 dictionary. We remark that the dictionary can be indexed off of the salt and the password verification value, and that once Mallory finds an HMAC-SHA1 key such that the MAC of the encrypted file verifies, he will with high probability learn the user's corresponding passphrase. While this is certainly a time-memory trade-off in terms of not having to compute PBKDF2 for every passphrase guess, the memory and precomputation requirements are still quite enormous and we expect that in practice anyone trying to learn a passphrase will simply try to exhaustively search the passphrase, rather than try to use an HMAC-SHA1 key dictionary.

8 Fixes

Although our description of the Zip file format in Section 2 is fairly simple, it is important to note that the Zip file format [9] is actually significantly more complex, allowing, for example, Zip utilities to use different date and time formats, to handle files larger than 2^{32} bytes, and to create multi-volume CD archives. While the added complexities to the Zip file format do not affect our attacks, they do make addressing the information leakage issues raised in Section 3 rather difficult. Therefore, although it is conceivable that a solution to the information leakage concerns raised in Section 3 may be possible in the future, albeit with intrusive changes to the Zip file format specification, it seems that addressing the issues that we raised in Section 3 while under the current Zip format umbrella may be unrealistic, at least in the short term and without compromising on some of WinZip Computing, Inc.'s current (non-security-related) goals, e.g., the aforementioned engineering, usability, and backward-compatibility issues. Consequently, we shall discuss how to address all of the other issues we raise, but leave the issues in Section 3 untouched.

Let us ignore chosen-protocol attacks for now. To address the problems raised in Section 4, one approach might be to MAC the original uncompressed plaintext instead of the ciphertext and then encrypt the resulting tag in a MAC-then-Encrypt-style construction. However, we do not recommend this as a general design procedure since the resulting construction may not be generically secure (cf., the counter examples for MAC-then-Encrypt in [1, 16]). Much better would be to build off of WinZip's current Encrypt-then-MAC core since Encrypt-then-MAC is known to be generically secure (again due to [1, 16]). Having decided to continue to use the existing Encrypt-then-MAC core, which is also attractive since reusing the current core means less modifications to the WinZip encryption specification, we note the following general design principle for cryptographic encapsulation methods: A cryptographic encapsulation algorithm should authenticate *all* of the information that an extractor/decapsulator will use when reconstructing the original data, excluding the authentication tag itself and assuming that the extractor already has a copy of the shared authentication key. In the case of WinZip, since the compression type field of an encrypted file's header will be accessed when extracting an encrypted file, this means that the compression type value should be MACed along with the AES-CTR-generated ciphertext. We can naturally extend this general principle to mandate the authentication of all data necessary to ensure the correct *interpretation* of the data once the data has been correctly reconstructed, which means that the file name and date fields in an encrypted file's header must also be authenticated, which addresses the concerns raised in Section 5.

To prevent chosen-protocol attacks, it might be tempting to apply the above principle and create a new AE version that MACs the encryption method version number field in the extra data field of an encrypted file's header. This, however, does not necessarily work since here we are concerned about attacks that exploit the interaction between different encapsulation/decapsulation schemes, and, in particular, interactions with schemes, AE-1 and AE-2, that have already been specified and that do not currently authenticate that field. To see why this is a problem, note that an adversary could move the extra data MACed using the new method into the ciphertext portion of an AE-2-format archive and thereby mount a chosen-protocol attack. While one might try MACing information not directly available to an adversary, we view such an approach as inelegant. Rather, we suggest diversifying the AES and HMAC-SHA1 key derivation process in such a way that the AES and HMAC-SHA1 keys derived from some passphrase and salt using the new encryption method will be different from the keys derived from the same passphrase and salt when using the AE-1 and AE-2 encryption methods. This could involve, for example, prepending the encryption method version number, vendor ID, and encryption strength field to the salt before running the key derivation procedure.

Finally, to address the issues raised in Section 7, we suggest two possible solutions. First, instead of always using zero as the initial AES-CTR mode counter, one could use a random initial counter selected from the set of all possible 128-bit integers. The initial counter should be included in the resulting archive and should also be included in the string to be MACed. Furthermore, under this approach the same AES and HMAC-SHA1 keys can be used with all files protected by the same passphrase; i.e., the same randomly-selected salt could be used with all such files in an archive. Alternatively, one could double the current salt length.

Complete details, as well as security definitions and generic security proofs, to appear in the full version of this paper.

References

[1] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer-Verlag, Berlin Germany, Dec. 2000.

[2] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer-Verlag, Berlin Germany, Dec. 2000.

[3] D. Benedetto, E. Caglioti, and V. Loreto. Language trees and Zipping. *Physical Review Letters*, 88(4), Jan. 2002.

[4] E. Biham. How to decrypt or even substitute DES-encrypted messages in 2^{28} steps. *Information Processing Letters*, 84, 2002.

[5] E. Biham and P. Kocher. A known plaintext attack on the PKZIP stream cipher. In B. Preneel, editor, *Fast Software Encryption ' 94*, volume 1008 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Germany, 1994.

[6] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, Berlin Germany, 2002.

[7] P. Deutsch. DEFLATE compressed data format specification version 1.3. IETF Request for Comments 1951, May 1996.

[8] S. R. Fluhrer and D. A. McGrew. Statistical analysis of the alleged RC4 keystream generator. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer-Verlag, Berlin Germany, Apr. 2000.

[9] Info-ZIP. Info-ZIP note, 20011203, Dec. 2001. Available at `ftp://ftp.info-zip.org/pub/infozip/doc/appnote-011203-iz.zip`.

[10] K. Jallad, J. Katz, and B. Schneier. Implementation of chosen-ciphertext attacks against PGP and GnuPG. In A. H. Chan and V. D. Gligor, editors, *Information Security, 5th International*

Conference, volume 2433 of *Lecture Notes in Computer Science*, pages 90–101. Springer-Verlag, Berlin Germany, 2002.

[11] D. W. Jones. *The Case of the Diebold FTP Site*, July 2003. Available at `http://www.cs.uiowa.edu/~jones/voting/dieboldftp.html`.

[12] B. Kaliski. PKCS #5: Password-based cryptography specification version 2.0. IETF Request for Comments 2898, Sept. 2000.

[13] J. Katz and B. Schneier. A chosen ciphertext attack against several e-mail encryption protocols. In *Ninth USENIX Security Symposium*, 2000.

[14] J. Kelsey. Compression and information leakage of plaintext. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 263–276. Springer-Verlag, Berlin Germany, 2002.

[15] J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *Security Protocols: 5th International Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104. Springer-Verlag, Berlin Germany, 1997.

[16] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer-Verlag, Berlin Germany, Aug. 2001.

[17] I. Mironov. (not so) random shuffles of RC4. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 304–319. Springer-Verlag, Berlin Germany, 2002.

[18] PKWARE. APPNOTE.TXT - .ZIP File Format Specification, Jan. 2004. Version 6.1.0, available at `http://www.pkware.com/products/enterprise/white_papers/appnote.txt`.

[19] P. Rogaway. Authenticated encryption with associated data. In V. Atluri, editor, *Proceedings of the 9th Conference on Computer and Communications Security*, Nov. 2002.

[20] L. Spector. WinZip, PKWare call truce in format war. *PC World*, Jan. 2004. Available at `http://www.pcworld.com/news/article/0,aid,114401,00.asp`.

[21] M. Stay. ZIP attacks with reduced known plaintext. In M. Matsui, editor, *Fast Software Encryption 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 124–134. Springer-Verlag, Berlin Germany, 2001.

[22] WinZip Computing, Inc. Advanced encryption technology, Mar. 2004. Available at `http://www.winzip.com/wzdaes.htm`.

[23] WinZip Computing, Inc. AES encryption information: Encryption specification AE-2, Jan. 2004. Version 1.02, available at `http://www.winzip.com/aes_info.htm`.

[24] WinZip Computing, Inc. Homepage, Mar. 2004. Available at `http://www.winzip.com/`.

[25] WinZip Computing, Inc. What's new in WinZip 9.0, Mar. 2004. Available at `http://www.winzip.com/whatsnew90.htm`.