IACR ePrint Archive 2004/078.

Analysis of the WinZip encryption method

Tadayoshi Kohno[*]

May 8, 2004

Abstract

WinZip is a popular compression utility for Microsoft Windows computers, the latest version of which is advertised as having "easy-to-use AES encryption to protect your sensitive data." We exhibit several attacks against WinZip's new encryption method, dubbed "AE-2" or "Advanced Encryption, version two." We then discuss secure alternatives. Since at a high level the underlying WinZip encryption method appears secure (the core is exactly Encrypt-then-Authenticate 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.

Keywords: WinZip, Zip, compression, encryption, applied cryptography, 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 [25] is a popular compression utility for Microsoft Windows computers, the latest version of which is advertised as having "easy-to-use AES encryption to protect your sensitive data" [25]. 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 personal data. Additionally, because of WinZip's Microsoft Outlook email plugin [24] and given other comments on WinZip's websites [25, 26], we fully anticipate that many users will also choose to use WinZip's encryption feature in an attempt to cryptographically protect the contents of their email attachments and other shared data.

Unfortunately, WinZip's latest encryption scheme, dubbed "Advanced Encryption-2" or AE-2 [23] and shipped with WinZip 9.0, is insecure in a number of natural scenarios. We exhibit several attacks in this paper and then propose ways of fixing the protocol. We believe that our proposed fixes to the Zip file format are relatively non-intrusive and that they will require only a moderate amount of reimplementation on the part of WinZip Computing, Inc. and the vendors of 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] A WinZip archive can contain multiple files, and when that is the case, each file is compressed and encrypted independently. For each file to archive, if the length of the file is above some threshold, WinZip first compresses the file using some standard compression method such as DEFLATE [8]. WinZip then invokes the AE-2 encryption method on the output of the previous stage. Specifically, it derives AES [7] and HMAC-SHA1 [17] keys from the user's passphrase and then encrypts the output of the compression stage with AES in counter (CTR) mode (AES-CTR) and authenticates 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 [17] and standard assumptions on AES-CTR and HMAC-SHA1. In other words, at a high-level, the new WinZip encryption architecture appears quite solid.

A collection of issues. All our attacks exercise different problems with the way that WinZip attempts to protect users' files. Furthermore, our attack works in a variety of different settings, require a variety of different resources, and accomplish a variety of different goals, 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. We believe that our observations also serve to highlight the subtlety of cryptographic design since (1) the WinZip AE-2 encryption method uses a provably-secure Encrypt-then-Authenticate core in a natural and seemingly secure way and (2) one of the attacks we discover was made possible because of the way that WinZip chose to fix a different problem with its earlier encryption method, AE-1. Furthermore, (1), as well as some of the other attacks that we discuss, underscore the fact that security products must be evaluated as a whole, and that the security of a whole product may not follow as a simple corollary of the security of some underlying component.

The main issues we uncover include the following:

[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."

Information leakage from encrypted files' metadata. According to the WinZip documentation, there is a known problem with the WinZip encryption architecture in that the metadata of an encrypted file appears in the WinZip archive in cleartext. Contained in this metadata is the encrypted file's original filename, 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 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 [15] shows that an adversary knowing only the length of an uncompressed data stream and the length of the compression output will be able to learn some information about the uncompressed data. 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 the entire `.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.

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. Bellare and Namprempre [1] and Krawczyk [17], an attacker can exploit the fact that the metadata 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 do this, they first agree upon a shared secret passphrase. Suppose Alice uses WinZip to compress and encrypt some file named `F.dat`, 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 using WinZip's Outlook email plugin 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 authenticated encryption [14, 2, 1]. and secure channel [6, 17] notions; i.e., the construction should preserve the privacy and the authenticity of Alice's files. 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 metadata 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 the contents of `F.dat` upon extraction will not be the original contents of `F.dat`, but will will now look 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 the true contents of Alice's original `F.dat` file.

We believe that the above attack scenario is quite realistic. In fact, it is the same scenario that Katz and Schneier [13] and Jallad, Katz, and Schneier [10] used when attacking email encryption programs and PGP, so any attack against WinZip's Outlook email plugin under the same scenario is at least as damaging (one difference is that our attack is applicable to WinZip in its default setting, whereas the previous attacks against PGP require the user to choose a non-default setting or to encrypt already compressed data). Even when users do not use WinZip's Outlook plugin to send encrypted attachments, we believe that there are other natural scenarios in which an adversary could mount our attack. For example, employees of at least one large corporation, Diebold Election Systems, transported important election-related files, compressed and encrypted into Zip archives, via an anonymous ftp site [11].[2] 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 output G. Lastly, even if security-conscious users might try to prevent an adversary from learning G, we believe that security products should remain secure even in the face of potential misuses by non-security conscious users, which further suggests that the attack we describe is significant and should be protected against.

On the names of files and their interpretations. There are a number of systems that associate software applications with filenames; 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 does not authenticate an encrypted file's filename metadata field, 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, 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 filename unencrypted; specifically, the issue is not that the filename is stored in cleartext, but that the filename is not authenticated, though also encrypting the filename would not hurt.

We discuss other issues that can arise from allowing an adversary to modify the names of encrypted files. The main lesson with all of these issues is that a file encryption utility must not only protect the integrity of the *contents* of an encrypted file, but must also protect the integrity of all of the *metadata*, like the filename or filename 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 also exploits the fact that in addition to using HMAC-SHA1, AE-1 also uses a 32-bit CRC of the unencrypted plaintext file.

The attack works in the same setting as the previous attacks. In this attack, Mallory intercepts `F.zip`, makes a guess of the contents of `F.dat`, and creates a replacement `F-prime.zip` based off his guess. If Bob can successfully decrypt `F-prime.zip`, i.e., if Bob doesn't complain to Alice that the file failed to decrypt because of a failed CRC check, then Mallory learns with high probability whether his guess 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 was correct. Still, this may constitute a serious attack if Mallory knows

[2] These events preceded WinZip's invention of AE-2 and Diebold used the traditional Zip encryption method.

that the contents of `F.dat` 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.)

Archives with both encrypted and unencrypted files. According to the WinZip AE-2 specification, archives can contain both encrypted and unencrypted files. While this may have some functionality and usability advantages, there is also a rather serious security disadvantage. In particular, when a user invokes WinZip 9.0's extraction utility on an archive containing both encrypted and unencrypted files, WinZip 9.0 will ask for a passphrase. It will then proceed to extract *all* of the files in the archive, without telling the user which files were encrypted and which were not. The user will thus think that all the files in the archive were encrypted (and authenticated), but, in fact, an adversary could have complete control over the contents of all but one of the files in the archive (one file must remain encrypted under the user's passphrase in order to force WinZip 9.0 to prompt the user for the passphrase). (In Section 7 we provide evidence that suggests that although WinZip Computing, Inc. was unaware of the attack we found when they designed AE-2, other Zip manufacturers may have been aware of it, or at least knew that there were risks associated with allowing both encrypted and unencrypted files in Zip archives.)

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 AES key collisions after encrypting only 2^{32} files when using AES with 128-bit keys. Furthermore, the AE-2 specification says that the initial CTR mode counter is always zero.[3] Combining these two observations, we can expect CTR mode 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.

Because WinZip encrypts each file in an archive independently, all 2^{32} 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 a 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. There are other ways in which an adversary might attack WinZip 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 new 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

[3] Previously we said that the underlying Encrypt-then-Authenticate core of AE-2 is a provably secure authenticated encryption scheme per Bellare and Namprempre [1] and Krawczyk [17]. 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.

asks for but 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 and WinZip's use of cryptography, we do not consider these attacks further.

Secure alternatives. In response to the cryptographic issues and attacks we found, we discuss a number of approaches for fixing the WinZip encryption method while simultaneously minimizing the changes to the AE-2 specification. Since WinZip Computing, Inc. has a business interest in competing with other companies offering archival and email programs with similar or better advertised security claims, like PKWARE, and since we believe that WinZip Computing, Inc. truly cares about protecting the privacy and integrity of users' data, we hope that WinZip Computing, Inc. will choose to incorporate many of our suggestions into their application.

Other Zip encryption methods. There are a number of other passphrase-based Zip encryption methods besides WinZip's new AE-2. The traditional Zip encryption mechanism [9] has similar functionality to AE-2, but it has significantly worse security: The traditional Zip stream cipher has been broken [5, 22] and the contents of traditionally-encrypted archives can be efficiently recovered from the archives directly; i.e., there is no need to mount a chosen-ciphertext attack like the ones we describe above. PKWARE also recently announced a new passphrase-based encryption mechanism called EFS [19]. The January 2004 version of the PKWARE's EFS specification [20], as well as the traditional Zip encryption mechanism, are all vulnerable to our attacks that exploit generic properties of the Zip file format, namely the attacks exploiting (1) the information leakage of an encrypted file's metadata, (2) the fact that an encrypted file's filename is not authenticated, and (3) the fact that an archive can contain both encrypted and unencrypted files. Although the global applicability of issue (1) is by now folklore knowledge, and we have evidence to believe that some people, although unfortunately not WinZip Computing, Inc., may have known about some aspects of issue (3), we have seen no previous discussions of issue (2). The lack of previous discussions and awareness of these latter issues is likely because, until the creation of applications like Zip Outlook plugins, and until the publication of works like Katz and Schneier [13], the risks of chosen-ciphertext attacks were under-estimated.

The latest EFS specification [19], dated April 26, 2004 and appearing after the original IACR ePrint appearance of this paper, adds a new "filename encryption" feature that will encrypt the filename and other metadata fields of encrypted files. Although EFS's approach for addressing issue (1) is different than ours, and is an option that users or administrators may fail to turn on (it was not the default in the version we tested), we are pleased to find that our suggestions for fixing (1) are less intrusive to the Zip file format than PKWARE's (when "filename encryption" is turned on under PKWARE's new specification [19], PKWARE-encrypted archives are not parsable under the traditional Zip specification [9]). With respect to PKWARE's specification's new "filename encryption" feature, we note that "filename encryption" alone cannot always fully protect against our problems with issues (2) and (3), largely because encryption alone does not necessarily imply authentication (we do remark, however, that the use of certain encryption modes and the compression of the central directory, which is how PKWARE achieves filename encryption, may prevent the attacks from immediately going through, but this is largely for fortuitous reasons). PKWARE's specification [19] also includes the ability to encrypt and sign files using public key cryptography, assuming the presence of the requisite additional infrastructure, though it is worth noting that the "certificate processing method for ZIP file encryption remains under development ... and is subject to change without notice [19]." Although a full treatment of PKWARE's new EFS passphrase-based encryption mechanism, as well as PKWARE's use of public key cryptography, is outside the scope of this paper, we do make a few observations here. The passphrase-based encryption mechanism does not include a message authentication code at all, and thus does not appear to have been designed to protect the privacy or integrity of files under chosen-ciphertext attacks. This is problematic since, although digital signatures can be used to protect the authenticity of the encapsulated data, it is still important to protect the authenticity of files encrypted with passphrases when the necessary infrastructure for digital signatures is not available, or when a user does not want to be bound to the contents of a file with a digital signature. The specification is also incomplete, making it not only difficult to implement the system from the specification alone, but to fully analyze the system for potential security problems without making conjectures about how the system is actually supposed to work; e.g., if the user or developer chooses RC4 for encryption, how exactly is RC4 supposed to be used? Are results like Mironov's [18] taken into consideration? Where the specification is unambiguous, the specification still leaves decisions, such as the choice of the underlying cipher (e.g., 40-bit RC2, 64-bit RC4, 3DES, AES) and the length of the randomness RD when deriving encryption keys, up to the choice of implementors. This is a concern since even if PKWARE makes safe choices with respect to these decisions, there is nothing in the specification to prevent third-party developers from making unsafe choices.

Additional related works. In addition to the already-cited 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 keystream reuse attack in Section 8 is related to Biham's key-collision attack except that it is more efficient than a normal key collision attack because of the way that WinZip derives AES keys from passphrases. Kelsey, Schneier, and Wagner [16] introduced the concept of a chosen-protocol attack.

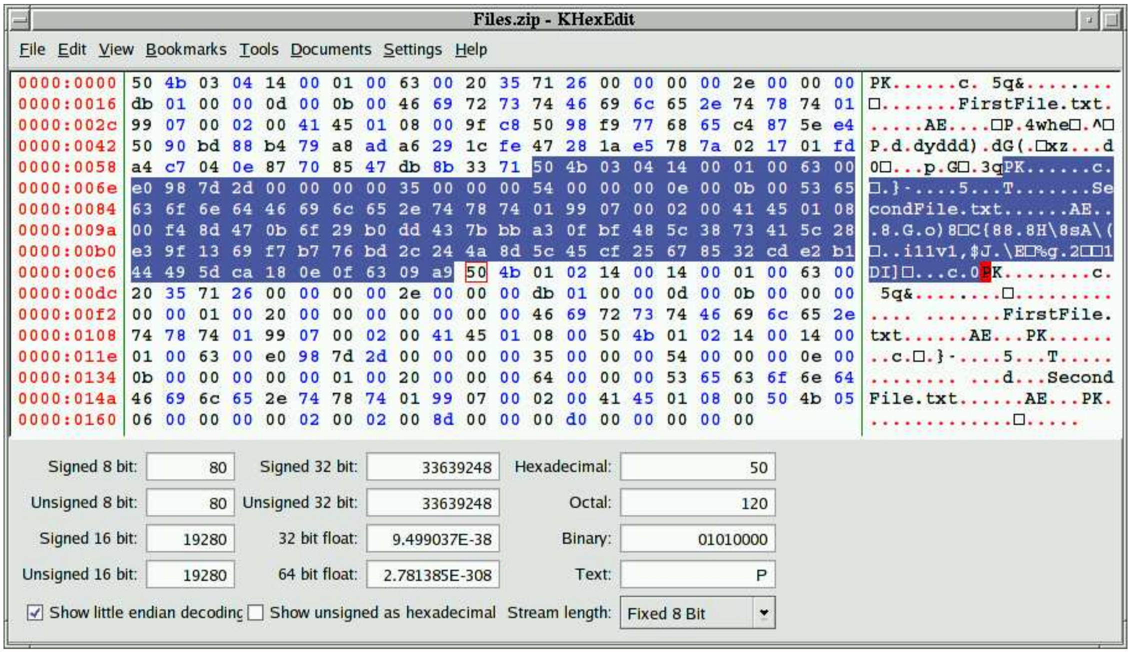
2 The WinZip compression and encryption method

WinZip's compression architecture follows the Info-ZIP specification [9]. The AES-based AE-2 extension is described on WinZip's website [23]. The difference between the AE-2 encryption method and the AE-1 encryption method is slight and will be mentioned at the end of this section.

Basic structure. We present here the basic Zip file format and the AE-2 extensions, omitting details that are not relevant to our attacks and to our security improvements. Figure 1 shows the contents of an example AE-2-encrypted WinZip archive.

A Zip archive can contain multiple files. When archiving a set of files, WinZip creates two *records* for each file, a *main file record* and a *central directory record*. The resulting Zip archive contains all of the main file records concatenated together followed by all of the central directory records (following the central directory records is an *end of archive record*, which is not relevant to our attacks and suggested improvements). The *main file record* contains metadata about the file, like the filename, as well as the file's contents, the latter typically being compressed and, in the case of AE-2, encrypted. The contents of each file is compressed and encrypted independently. The *central directory record* mirrors the metadata stored in the main file record and also contains information about the location of the file's corresponding main file record in the Zip archive. One of the reasons for the existence of the central directory record is for usability when working with multi-volume floppy or CD archives. For example, when extracting a file from a multi-volume CD archive, the user can insert the last CD, WinZip can read the central directory information, and then WinZip can prompt the user to insert the CD containing the main file record.

When referring to fields of Zip archive, 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.



```
0000:0000 50 4b 03 04 14 00 01 00 63 00 20 35 71 26 00 00 00 00 2e 00 00 00 PK......c. 5q&........
0000:0016 db 01 00 00 0d 00 0b 00 46 69 72 73 74 46 69 6c 65 2e 74 78 74 01 □.......FirstFile.txt.
0000:002c 99 07 00 02 00 41 45 01 08 00 9f c8 50 98 f9 77 68 65 c4 87 5e e4 .....AE....□P.4whe□.^□
0000:0042 50 90 bd 88 b4 79 a8 ad a6 29 1c fe 47 28 1a e5 78 7a 02 17 01 fd P.d.dyddd).dG(.□xz...d
0000:0058 a4 c7 04 0e 87 70 85 47 db 8b 33 71 50 4b 03 04 14 00 01 00 63 00 0□...p.G□.3qPK......c.
0000:006e e0 98 7d 2d 00 00 00 00 35 00 00 00 54 00 00 00 0e 00 0b 00 53 65 □.}-....5...T.......Se
0000:0084 63 6f 6e 64 46 69 6c 65 2e 74 78 74 01 99 07 00 02 00 41 45 01 08 condFile.txt......AE..
0000:009a 00 f4 8d 47 0b 6f 29 b0 dd 43 7b bb a3 0f bf 48 5c 38 73 41 5c 28 .8.G.o)8□C{88.8H\8sA\(
0000:00b0 e3 9f 13 69 f7 b7 76 bd 2c 24 4a 8d 5c 45 cf 25 67 85 32 cd e2 b1 □..i11v1,$J.\E□%g.2□□1
0000:00c6 44 49 5d ca 18 0e 0f 63 09 a9 50 4b 01 02 14 00 14 00 01 00 63 00 DI]□...c.0PK.........c.
0000:00dc 20 35 71 26 00 00 00 00 2e 00 00 00 db 01 00 00 0d 00 0b 00 00 00  5q&........□.........
0000:00f2 00 00 01 00 20 00 00 00 00 00 00 00 46 69 72 73 74 46 69 6c 65 2e .... ........FirstFile.
0000:0108 74 78 74 01 99 07 00 02 00 41 45 01 08 00 50 4b 01 02 14 00 14 00 txt.......AE...PK......
0000:011e 01 00 63 00 e0 98 7d 2d 00 00 00 00 35 00 00 00 54 00 00 00 0e 00 ..c.□.}-.....5...T.....
0000:0134 0b 00 00 00 00 00 01 00 20 00 00 00 64 00 00 00 53 65 63 6f 6e 64 ......... ...d...Second
0000:014a 46 69 6c 65 2e 74 78 74 01 99 07 00 02 00 41 45 01 08 00 50 4b 05 File.txt.......AE...PK.
0000:0160 06 00 00 00 00 02 00 02 00 8d 00 00 00 d0 00 00 00 00 00             ..............□.....
```

Signed 8 bit: 80 Signed 32 bit: 33639248 Hexadecimal: 50

Unsigned 8 bit: 80 Unsigned 32 bit: 33639248 Octal: 120

Signed 16 bit: 19280 32 bit float: 9.499037E-38 Binary: 01010000

Unsigned 16 bit: 19280 64 bit float: 2.781385E-308 Text: P

☑ Show little endian decoding □ Show unsigned as hexadecimal Stream length: Fixed 8 Bit

Fig. 1. This figure shows the contents of a Zip archive containing two files, `FirstFile.txt` and `SecondFile.txt`, both compressed and encrypted using the WinZip AE-2 encryption method. The highlighted portion corresponds to the main file record for `SecondFile.txt`.

Main file record. According to the Info-ZIP specification [9], and barring certain extensions that do not affect our attacks, all main file records have the following structure (the fields important to our work are highlighted): main file record indicator (4 bytes, always `504b0304`$_{\text{bs}}$), 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), filename length (2 bytes), extra field length (2 bytes), *filename* (variable size), and *extra field* (variable size). Following the above fields, but still part of the main file record, is the *file data* field.

Central directory record. The central directory record for a file consists of the following fields (important fields highlighted): central directory record indicator (4 bytes, always `504b0102`$_{\text{bs}}$), version made by (2 bytes), 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), filename length (2 bytes), extra field length (2 bytes), file comment length (2 bytes), disk number start (2 bytes), internal file attributes (2 bytes), external file attributes (4 bytes), relative offset of local header (4 bytes), *filename* (variable size), *extra field* (variable size), and file comment (variable size).

AE-2 settings and the AE-2 extra data field. The following is applicable to both the main file record and the central directory record. 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 method are set to `6300`$_{\text{bs}}$. The extra data field will consist of the following 11 bytes (again, important fields highlighted): extra field header id (2 bytes, always `0199`$_{\text{bs}}$), data size (2 bytes, `0700`$_{\text{bs}}$ for AE-2 since there are seven remaining bytes in the 11-byte extra data field), *version number* (2 bytes, always `0200`$_{\text{bs}}$ for AE-2), 2-character vendor

ID (2 bytes, always $\texttt{4145}_{\mathrm{bs}}$ 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 $\texttt{01}_{\mathrm{bs}}$ (resp., $\texttt{02}_{\mathrm{bs}}$ or $\texttt{03}_{\mathrm{bs}}$) 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 $\texttt{0800}_{\mathrm{bs}}$ if the file is DEFLATEd [8] and $\texttt{0000}_{\mathrm{bs}}$ if no compression is used.

File data field. When a file is AE-2-encrypted, the file data field of the main file record contains the following information: *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.

The encrypted file data and the authentication code. Before applying the AE-2 encryption method, the contents of the plaintext file is compressed according to the "actual compression method used to compress the file" field of the AE-2 extra data field described above. 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 described 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.

Differences between AE-1 and AE-2. The only differences between the AE-2 method and the earlier AE-1 method is that in AE-1 the version number in the main file record's and central directory record's extra data fields are $\texttt{0100}_{\mathrm{bs}}$ and the 32-bit CRC fields are not all zero but actually contains the CRC of the original unencrypted data, which the WinZip specification [23] states 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 Information leakage

The metadata fields of encrypted files leak important and potentially security-critical information in several ways. The names of the encrypted files are stored in cleartext, which can obviously be a concern. The files' last modification dates and times are also stored unencrypted, which can be used to infer some relationship between the contents of different encrypted files or some event in the past. Additionally, the length of plaintext files are stored in the files' metadata fields unencrypted. This is a concern since, based on Kelsey's recent results about compression as a side-channel [15], 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 file's metadata, but the risks associated with leaving these fields unencrypted is not considered. Furthermore, many users may fail to read the documentation, and thus not realize that these information leakage side-channels exist in the first place.

It is a well known fact that the classic Zip encryption method [9] also leaks the information that we mention above, plus the 32-bit CRC of an encrypted file's original plaintext. It is interesting to ask why WinZip Computing, Inc. did not fix this problem in their new AE-2 specification. The most likely conjecture is that WinZip Computing, Inc. chose not to do so either because of engineering or design complexities, or because of functionality issues (e.g., they actually wanted to allow users to be get a directory listing of the contents in their encrypted archives without having to enter a passphrase). To address the former reason, we discuss technical approaches for addressing the information leakage concerns in Section 10.

4 Exploiting the interaction between compression and encryption

Recall the setup described in Section 1, where Alice encrypts `F.dat` and sends the resulting Zip archive, `F.zip`, to Bob, but where Mallory prevents the delivery of `F.zip` and instead gives Bob a file, `F-prime.zip`, that is related to `F.zip` but that is slightly different. The critical observation for our attack is that despite the fact that the underlying encryption core is a provably secure Encrypt-then-Authenticate authenticated encryption scheme, cf. [1, 17], the compression method and original file length fields in an encrypted file's main file and central directory records are *not* authenticated, which means that an adversary can change these fields 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 method, the file will be decompressed using the wrong algorithm and the resulting contents G of the extracted file will look like garbage. If Mallory can learn G, which we argue in Section 1 is reasonable in some cases, Mallory can recover the original contents of Alice's file `F.dat`.

Implementing the attack. When actually mounting the attack, Mallory would probably change the compression method indicators in the main file and central directory records from $\texttt{0800}_{\text{bs}}$, which appears to be WinZip's default and which corresponds the DEFLATE algorithm [8], to $\texttt{0000}_{\text{bs}}$, which corresponds to no compression. This is very easy to do and very efficient and can be done in a linear pass through the file, as can updating the original file length field. 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 $\texttt{0800}_{\text{bs}}$ to $\texttt{0000}_{\text{bs}}$, 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.[4] We would only expect the above command line to not work as desired if the 7-byte string $\texttt{02004145010800}_{\text{bs}}$ appears in `F.tar` in a place not corresponding to the extra data field of a file's main file or central directory records. 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 original file length fields of the file's main file and central directory records.

Subtlety of cryptographic design. 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 data G will not be saved and the attack will not immediately go through. While

[4] Different versions of `sed` appear to handle binary streams differently. The attack worked on default RedHat 9.0 systems with `sed` version 4.0.3.

it is true that if Bob is crafty he may be able to view `F.dat` (the file with contents 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 `F.dat` 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 filenames

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' filename extensions; e.g., Microsoft Windows will by default open `.doc` files with Microsoft Word. Since the filename fields of an encrypted file's main file and central directory records are unauthenticated, an adversary could modify those field without voiding the MAC included at the end of the encrypted file's main file record. Once Mallory does this, he can mount a variant of the attack in Section 4 since applications will usually report an error when trying to open a file of the wrong extension. Fortunately, some applications give descriptive error messages and, and Bob may realize that the file has the wrong filename extension (e.g., Microsoft Excel gives the error "File.xls: file format is not valid" when opening a document created with Microsoft Word), but this is largely serendipitous and should not be relied upon for security. This discussion confirms the fact that a file encryption utility must not only protect the integrity of the encapsulated data itself, but also the metadata, like the filename extension, necessary for the surrounding system to correctly interpret that data.

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 file is included unencrypted in an AE-1-encrypted WinZip archive, and that will leak information about the encrypted files' contents. While the CRC is no longer included in the output of the AE-2 encryption method, one can exploit an 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. Our attack makes use of the fact 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.

Details. Recall the setting used in Section 4 and Section 5. 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 H. He can do this as follows: Compute the 32-bit CRC of H and then modify `F.zip` such that the version number in the main file and central directory records' extra data fields are `0100`$_{\mathrm{bs}}$ and the CRC fields in the file's main file and central directory records has the CRC of H. 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.

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, if so, will likely convey this lack of confidence to Alice. Other implementations of the AE-2 specification may delete the extracted file before informing the user that the CRC check failed.

Extensions. 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 the plaintext file, and thereby learn additional information about the plaintext.

7 Attacking Zip encryption at the file level

When a Zip archive contains multiple files, each of the files in the archive is encapsulated independently, which means that some files in an archive may only be compressed and some may be both compressed and encrypted. Unfortunately, this functionality also opens the WinZip encryption method to attack.

This fact makes the WinZip AE-2 encryption method vulnerable to a number of attacks. For example, consider the following: Mallory knows that the encrypted archive `Salaries.zip` contains the files `Alice-Salary.dat`, `Bob-Salary.dat` and `Mallory-Salary.dat`, all encrypted using AE-2 under the CFO's secret passphrase. Now, because of the properties described above, an adversary could remove the encrypted `Mallory-Salary.dat` file from the `Salaries.zip` archive and replace it with a *new*, *unencrypted* file, also named `Mallory-Salary.dat`, but with the contents of Mallory's choice. When the CFO tries to extract the files in the archive using the WinZip 9.0 application, he will be prompted for his passphrase since the `Alice-Salary.dat` and `Bob-Salary.dat` files are still encrypted. WinZip will then extract the files `Alice-Salary.dat`, `Bob-Salary.dat`, *and* `Mallory-Salary.dat`. Since the CFO had to enter his passphrase, he will likely believe that the extracted `Mallory-Salary.dat` file is the same one that he encrypted, and thus contains Mallory's real salary, when in fact the contents of `Mallory-Salary.dat` are completely under Mallory's control. Similarly, if Alice creates an archive containing both encrypted and unencrypted files and sends that archive `F.zip` to Bob, Mallory will be able to easily modify the contents of the unencrypted files in the archive. But, like in the previous attack, since Bob has to enter a passphrase to extract the contents of the archive, and because no warning is given about some files being unencrypted, Bob will believe that all the files were encrypted by Alice and that they contain Alice's original content.

WinZip Computing, Inc. does not appear to have been aware of the above attacks when they specified AE-2 [23] and when they implemented WinZip 9.0, as supported both by the fact that WinZip 9.0 does not generate a warning when extracting an archive containing both encrypted and unencrypted files, and by the following quotes taken from the AE-2 specification [23], which only mention usability reasons for encrypting all the files in an archive and which does not suggest that vendors issue warnings when encountering unencrypted files in an archive with encrypted files:

> "[Section IV.A.] The presence of both encrypted and unencrypted files in a Zip [archive] may trigger user warnings in some Zip file utilities, so the user experience may be improved if all files (including zero-length files) are encrypted. Again, however, this is only a recommendation [23]."
>
> "[Section IV.B.] There is no requirement that all files in a Zip [archive] be encrypted or that all files that are encrypted use the same encryption method or the same password [23]."

The first quote does suggest, however, that other Zip vendors may have known of the attack we describe above, or at least knew to be wary of archives containing both encrypted and unencrypted files.

Because files in a Zip archive are encrypted on a per-file basis, an adversary could also delete files from an archive. An adversary could also create a composite Zip archive with encrypted files taken from multiple different archives, but we view these properties as less interesting than the first attacks in this section. Related to the first attacks in this section, in Section 5 we observed that an adversary could swap the filenames of different encrypted files, and that he could also use this fact to modify the contents of Alice's encrypted files; the attacks in Section 5 exploit a different security problem, that for encrypted files the filenames are not authenticated.

8 Keystream reuse

When AE-2 is used with a 128-bit AES key, one can expect CTR mode keystream reuse after encrypting approximately 2^{32} files, which is much less than one would expect given that AES has 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, and therefore we can expect the AE-2 encryption algorithm with 128-bit AES keys to start leaking even more information about the compressed and encrypted plaintext after 2^{32} files are encrypted with the same passphrase.

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.

9 Dictionary attacks

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 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. The dictionary can be indexed off of the salt *and* the two-byte password verification value; the password verification value thus reduces further reduces the amount of HMAC-SHA1 keys the attacker has to try in the birthday attack. 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, and thereafter be able to decrypt all of the files encrypted under that passphrase. While this is 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.

10 Fixes

In this section we consider fixes to the problems we discussed in Section 3 through Section 9, starting with Sections 4–9 and returning to Section 3 at the end. We also discuss our preferred instantiations of our suggestions.

We begin by ignoring chosen-protocol attacks. 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 Authenticate-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 Authenticate-then-Encrypt in [1, 17]). Much better would be to build off of WinZip's current Encrypt-then-Authenticate core since Encrypt-then-Authenticate is known to be generically secure (again due to [1, 17]). Having decided to continue to use the existing Encrypt-then-Authenticate core, 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 filename, date, and any other important metadata fields in an encrypted file's header must also be authenticated, which addresses the concerns raised in Section 5. (If WinZip Computing, Inc. does not mind deviating further from their current AES-CTR-then-HMAC-SHA1 construction, we note that the new encryption core can actually be any provably-secure *authenticated encryption associated data* scheme [21] as long as the important metadata fields are authenticated.)

To prevent chosen-protocol attacks like the one described in Section 6, 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. Unfortunately, this 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, such as the encipherment of some nonce, 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. If it were not the case that the length of the salt for AE-1 and AE-2 were fixed, but if the length of the salt was variable and if the length of the salt is encoded in a metadata field of an encrypted file, then even our solution here would not be a sufficient since an adversary could simply add the method version number, vendor ID, and encryption strength field into the (now larger) salt in an AE-2-formatted archive. For similar reasons, there is still the potential of interaction with other (non-WinZip) applications that uses PBKDF2-HMAC-SHA1, but it seems impossible for WinZip to complete avoid such interactions with applications that are not under their control.

There are several possible solutions for the problems raised in Section 7. The obvious approach of authenticating an entire archive would likely break some of WinZip Computing, Inc.'s functionality design criteria, namely the desire to (efficiently) handle updates to large archives, and in particular archives spanning multiple CD volumes. Another approach might be to authenticate the entire central directory (the concatenation of all the central directory records), since the central directory will always be stored at the end of the archive, and in particular on the last CD in a multi-volume archive. Toward this end, we note that the Zip specification already has the ability to sign the central directory using public key cryptography, so adding the ability to authenticate the central directory using a MAC is certainly reasonable. However, we point out that this solution has a number of issues that one must be careful of. For example, the extractor must check the consistency between the metadata in a file's main file record and a file's central directory record. If we are concerned about adversaries deleting files from an archive, then the absence of files must also be checked (this may follow as a corollary of checking the consistency of the individual files if the consistency check includes main file record offsets, which are stored in the central directory record). But of most concern is the fact that authenticating the central directory alone will *not* prevent an attacker from modifying unencrypted files in an archive. Rather, those unencrypted files must be cryptographically bound to the central directory in some way, perhaps by including a MAC of an unencrypted files in its central directory record. Another potential problem with this solution is that *if* authenticating the central directory is an option, then one must be careful to ensure that an adversary cannot simply take a Zip archive, turn that option off, and remove the MAC of the central directory. One possible way of handling this might be to use different AES and HMAC-SHA1 keys for when the option is turned on and when the option is turned off. But in reality, a reasonable solution might simply be to *require* applications implementing the AE-2 decryption algorithm to *always* report a warning when an archive contains both encrypted and unencrypted files.

To address the issues raised in Section 8, we suggest two possible solutions. First, one could double the current salt length. Alternatively, 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. The latter property is a performance win since in the current design, where a different salt is used with each file, the passphrase-based key derivation step dominates the time when creating or extracting archives containing lots of small files. Possible solutions to the issues raised in Section 9 include increasing the length of the salt or using the same salt when encrypting multiple files. Fortunately, these two recommendations align with our recommendations for the issues raised in Section 8. Additionally, we suggest not storing the password verification values in a file's metadata since it can be used to quickly eliminate keys in a dictionary attack against a user's passphrase.

There are a number of different approaches for addressing the information leakage concerns raised in Section 3. The latest (April 26, 2004) specification from PKWARE [19], which is incompatible with WinZip's new encryption method, introduces an option for encrypting the metadata fields of an encrypted file; when the option is turned on (it is not on by default), PKWARE's SecureZIP product encrypts the entire central directory and removes most of the metadata information from a file's main file record, either by zeroing out the appropriate fields or replacing them with random data. Aside from the fact that the central directory is not MACed, our two biggest issues with PKWARE's solution is that (1) we believe that protecting against information leakage from an encrypted file's header should not be an option and (2) archives created with the above option turned on are no longer parsable under the traditional Zip specification [9]. In contrast, our proposed fixes involve modifying the main file and central directory records such that privacy-critical metadata information is always hidden and the resulting Zip archives are still parsable under the traditional Zip specification [9]. We can achieve this goal in several ways. For example, using AES in CTR mode, it would be possible to encrypt specific metadata fields of a file's main file record and central directory record in-place. In the case of the central directory record, this approach would require us to copy the salt necessary to derive the encryption key from the file data field of the main file record into the extra data field of the central directory record. Unfortunately, this solution must still leak the length of a file's filename since, under this approach, we cannot encrypt any information necessary for parsing the file, and the length of a file's filename is necessary information. Consequently, the solution that we prefer is to not encrypt portions a file's main file record and central directory records in-place, but to encrypt (and also authenticate) the main file record and the central directory record completely. Our solution would then store the resulting ciphertext in the file data or extra data fields of a wrapper main file record or wrapper central directory record, respectively. Preceding the ciphertexts must be the information, like the salt, necessary to derive the file's cryptographic keys from the user's passphrase. The metadata fields of these wrapper records can be fixed, or random, as long as the "compression method field" in the main file record indicates that the record is just serving as a wrapper for an encrypted file. When extracting an archive, the extractor should see this specific compression method type, decrypt the wrapped data, and then treat the resulting plaintext as an unencrypted record to parse as normal. In order to give an intuitive error message to users who try to decrypt a file encrypted under this method, we suggest making the filename field of the wrapper records something like `WinZipEncryptedFile`; one could even add more information, like a URL. Lastly, another attractive property of this solution is that, by also authenticating these records completely, this solution immediately implements our previous recommendations for addressing the concerns in Section 4 and Section 5.

A POSSIBLE INSTANTIATION. Given the recommendations made in the above paragraphs, one possible instantiation might be the following, which is based on AE-2 but which we call BE since it is different enough to warrant a new name. For each file to archive, compress the file and create main file and central directory records as if encryption was not used. Then select a random value the same length as the salt in AE-2, concatenate information about the encryption scheme (BE algorithm identifier, version number, and AES-key-length value) with the random value, and call the resulting value the salt for BE. Derive AES and HMAC-SHA1 keys from the user's passphrase and the salt using PBKDF2-HMAC-SHA1. Then use that AES key to CTR mode encrypt all of the main file and central directory records, using a randomly selected initial counter (IV) for each record (the main file and the central directory records for a single file should have different random IVs). Then MAC the IVs concatenated with each of the ciphertexts using HMAC-SHA1. Then concatenate the BE algorithm identifier, version number, AES-key-length, the random value in the salt, the CTR mode IV, the ciphertext, and the MAC for each record. No password verification value is stored in these resulting strings. For the resulting string consisting of the encryption of the main file record, load it into the data portion of a wrapper main file record that has bit 0 of the general purpose flag set to 1 (meaning that the file is encrypted) and that has a "compression method" field indicating that the file is encrypted under our new encryption method; the other fields can by anything that does not leak information about the wrapped file. For the resulting string consisting of the central directory record, load it into the extra data portion of a wrapper central directory record that has the same general purpose flag and compression method as for the wrapper main file record. When extracting an archive, the user must be warned whenever encountering an unencrypted file in an archive with encrypted files. The MAC must also be checked during decryption. (Although all the data necessary to reconstruct a file is stored in the file's wrapped main file record, we still maintain the central directory record since it is part of the classic Zip file format [9] and since it will be used by some parties to quickly find specific files in an archive. If there are inconsistencies between a file's pair of records, an error should occur.)

Some caveats with the design. Although the same random value in the salt can be used for multiple files when encrypting them all at once, a new random value should be chosen if the user decides to update a file or add a new file to an archive. Alternatively, when updating a file or adding a new file to an archive, if one wants to use the same random value in the salt as before, they must check that the user's passphrase combined with the existing salts successfully decrypts currently-encrypted files. If either of these solutions were not in place, then an adversary could replace the random values in the salts in an archive with any value of his choice, and create a dictionary of AES and HMAC-SHA1 keys corresponding to the single chosen salt value. Additionally, when changing the contents of the file, and to avoid keystream reuse, a new random initial counter for CTR mode must be selected.

The security of this construction follows from the earlier discussions in this section and the provable security of AES-CTR-then-HMAC-SHA1 (unlike with AE-2, we can actually employ Bellare and Namprempre's [1] and Krawczyk's [17] results on the generic Encrypt-then-Authenticate paradigm when discussing BE since we are now encrypting *all* the data of interest, rather than just a portion of it). The risks associated to AES key collision attacks are minimized by the use of a random IV in AES-CTR (specifically, AES key collisions no longer immediately imply keystream reuse). BE can still leak information from the compression ratio of a file if the adversary knows the original length of the file (the original length is now no longer visible directly from the archive itself); this is acceptable because we are unaware of any solution to the information-leakage-through-compression problem without adding additional padding and thereby reducing the space savings generally associated to compression. Our new method is more efficient than AE-2 when adding multiple files to an archive in batch, or extracting multiple archives from a file in batch; this is because PBKDF2 is intentionally slow by design and, unlike AE-2, BE only invokes PBKDF2 once for all files added to an archive at the same time.

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] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 451–472. Springer-Verlag, Berlin Germany, 2001.

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

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

[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. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer-Verlag, Berlin Germany, Apr. 2000.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] PKWARE. APPNOTE.TXT - .ZIP File Format Specification, Apr. 2004. Version 6.2.0, available at `http://www.pkware.com/products/enterprise/white_papers/appnote.txt`.

[20] PKWARE. APPNOTE.TXT - .ZIP File Format Specification, Jan. 2004. Version 6.1.0, replaced by [19].

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

[22] 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.

[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. Download WinZip add-ons, Apr. 2004. Available at `http://www.winzip.com/daddons.htm`.

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

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