Crypto

Password Encryption and
Text Compression

By Mike Westerfield

TABLE OF CONTENTS

Introduction to Crypto

Crypto is a utility to help you protect disk files from prying eyes and put more text files on a disk than the disk would normally hold. If you are familiar with password encryption routines and text compression, and feel comfortable with the DOS or ProDOS operating system, you may not need to read this manual at all. But before you toss it in the desk, glance through it to see if any of the material is interesting to you.

How Encryption Programs Work

If you are still reading, you must want to know more - lets start by taking a look at what it means to encrypt a file. Knowing how the program works, at least at an overview level, will help you to understand the operation of the program a lot better than if you have no idea what is going on, and it is also fun to learn some trivia! On the other hand, it is not actually necessary to know about the internal workings, so you can skip the rest of the introduction if it seems too technical.

Almost all encryption programs use some variation of a code wheel hooked into a random number generator. You may have used a simple code wheel back in grade school when you passed notes to your friends. Figure 1 shows one that works for alphabetical characters only. The basic idea is to set the wheel to some predetermined place, such as the one shown with A and C matched up. You then go through the message to be encrypted, reading the letter in the message and matching it to a letter on the inside wheel, then write down the letter on the outside wheel. The result is a mess of almost incomprehensible letters that mean nothing until they are decoded by matching the letters in the coded message to the outside code wheel, then writing down the result from the inside code wheel. Incidentally, this kind of encryption is called a chipper, not a code.

Of course, it doesn't take too much time on a computer to break such a simple chipper. The next step is to rotate the code wheel for each letter. Choosing the position to rotate the code wheel to is the true heart of a modern chipper. Imagine for a moment that the code wheel is rotated for each letter to a completely random position. Then there woul,d be no pattern for anyone to detect, and no way for them to break the chipper! Unfortunately, there would also be no way to decode the message. The solution is to use a pseudo-random number generator, which is exactly the kind used by most computers. These random number generators don't really produce random numbers. Instead, they start with a random number seed, and produce a sequence of numbers which, hopefully, do not show a recognizable pattern. These numbers are then used to choose the setting of the code wheel for each of the letters in a message. Since the random number generator will produce the same sequence of numbers given the same starting seed number, anyone who has the seed that was used to encrypt the file can also decrypt it, but if the random number generator is a good one, no one else can make any sense of the file. This is the way that Crypto works, in a nutshell.

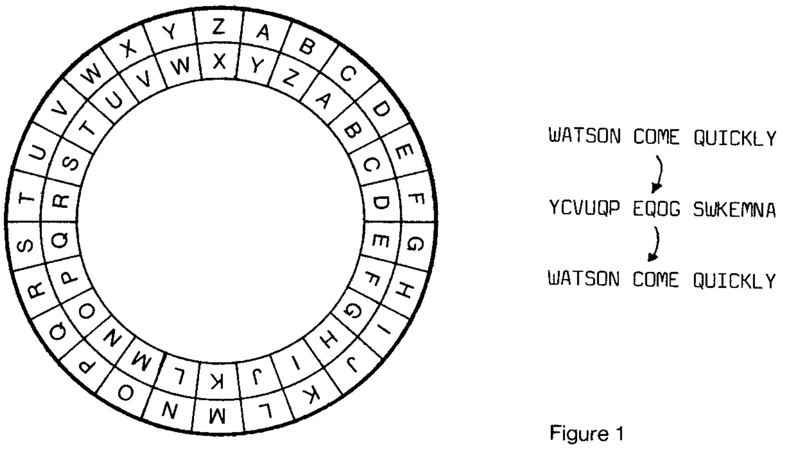


Figure 1

If you are a BASIC programmer, you may realize that by using the RND function in BASIC, you could write your own version of Crypto. Although this is true, and certainly a lot of fun, be aware that the result will not be as good as Crypto. The random number generator used in Applesoft, while good for simple tasks like choosing alternatives in a game, is not good enough for encrypting files. Part of the reason is connected to the fact that the Applesoft random number generator fails some of the tests for randomness that are used to test random number generators. This means that there is a pattern to the numbers, and the encrypted file can be broken. Also, the Applesoft random number generator uses only a two byte number between steps, which means that it could never generate more than 65536 numbers before it started to repeat, and is likely to generate far fewer. CRYPTO, on the other hand, uses sixteen bytes between steps. Theoretically, then, Crypto could generate as many as 3.4 x IOE38 numbers before it repeats far more than would be necessary to encrypt every book ever printed! Of course, in practice, it will repeat far quicker. How quick depends on the password used. If you need to encrypt any more than a city library, give us a call and we'll make the period longer.

How Text Compression Works

Crypto's tricks are not limited to encrypting your files; it can also make many of them smaller. The way this is done is based on a familiar fact - some letters in the English language occur far more frequently than others. In fact, if you look at the ASCII character set that your Apple can represent internally. you will find that may of the characters are almost never used. Yet, all characters require eight bits of space in your files. What if only two or three bits were used for the more common characters, while several bits, maybe even more than eight, were used for less frequently occurring characters? It turns out that if this is done right, English language text can be stored in a little over half of the disk space that it would normally occupy! Of course, the file must be converted back to its original form before being used, so that method is not very good for a file that is being used all the time. Its strength shows up when files that are not used very often, but which take up a lot of disk space, are compressed.

The actual method used to choose the bit patterns for each of the letters is called Huffman's Algorithm. It is a bit technical, so it won't be covered here, but if you are curious, check any advanced computer science text, or back issues of BYTE magazine for more information.

DOS or ProDOS

There are two popular operating systems available for the Apple, DOS and ProDOS. CRYPTO is supplied in two versions, one for each of these operating systems. If you put your disk into the drive in the normal way, the DOS version of Crypto will boot and run; to get the ProDOS version, turn the disk over and reboot. Which version you need will depend on which operating system you use.

Both versions work the same way, with the minor exception that with DOS you will select a slot and drive number, while with ProDOS you select a path name. Both are unprotected and copyable with any disk copy program. Please back up your disk and store the original in a safe place before going on. The original disk should only be put in a disk drive to copy it to another disk, and the copy should then be used for all future work.

Lower-case Displays

As it is shipped to you, Crypto uses all upper-case letters. Although this is necessary for older Apple][+ computers which do not have lower-case characters, if you have an Apple //e or Apple //c, you should take advantage of the second version of Crypto.

The only difference between the two if that one version uses lower-case and upper-case letters for output, while the other uses upper-case only.

To make the change, boot the Crypto disk and use Q to exit from the program. Now type the following commands, ending each line by hitting the RETURN key:

```
RENAME CRYPTO,CRYPTO.U

RENAME CRYPTO.L,CRYPTO

RUN CRYPTO
```

When you reboot the disk, the display should show both upper-case and lower-case letters. You must reboot for this to work, since Crypto gets upper-case only displays by installing a short machine language routine into the computer. This routine does not go away when Crypto stops.

Help Menus

The next-to-last option on the main menu is <H>, for help. This command gives you access to a short, one screen reminder that tells what each command is for. For the help command to work, you must have the help files in slot 6. drive I (or on the default prefix for ProDOS). If Crypto cannot find the help files, it will display an error message. The help files are all of the text files that start with the characters HELP. They are not copy protected, so programs like Apple Computer's FID, or their FILER, supplied for ProDOS, can be used to copy the help files to some other location.

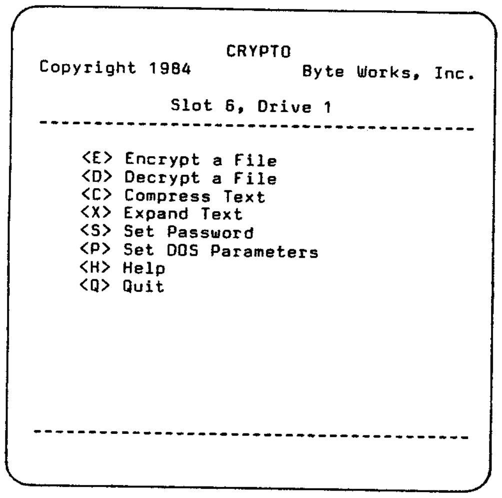
Encrypting Files

In this section, we will take a detailed look at the steps that you will have to go through to encrypt a file, as well as looking at the limits and exact capabilities of this part of the program. To start, place the Crypto disk in your disk drive and turn on the computer. If you are already in DOS, you can get started by simply typing

```
RUN CRYPTO
```

This will execute the Crypto program, which is written in Applesoft BASIC. The actual encryption and text compression subroutines are in the file CRYPTO.OBJ, which must be on the same disk as CRYPTO for the program to work.

Once Crypto comes up, your screen will look something like the one shown in Figure 2. The slot and drive numbers can change, or they could be path names if you are using the ProDOS version of Crypto.



```
                 CRYPTO
Copyright 1984          Byte Works, Inc.

            Slot 6, Drive 1
----------------------------------------

    <E> Encrypt a File
    <D> Decrypt a File
    <C> Compress Text
    <X> Expand Text
    <S> Set Password
    <P> Set DOS Parameters
    <H> Help
    <Q> Quit







----------------------------------------
```

Figure 2

Selecting a Password

The first thing that you will need to do is to set the password. This password becomes the key to the encrypted file. Without it, no one can read your file. Keep in mind that this includes you - loosing a password is as good as loosing the file itself! This should tell you right away that you must choose a password that you can remember. or that you should write the password down. On the other hand, it is not a good idea to use a password that another person might guess. A thief will commonly try things like your middle name, spouse or children's names, your birthday, etc.

What are the restrictions on a password? Well, it cannot be any longer than sixteen characters, and cannot include a RETURN key. Other than that, it can be anything you can type from the keyboard. The length of the password is easy to remember, since the password entry routine doesn't let you type more than sixteen characters, anyway.

After deciding what password to use, type P. You will be prompted for the password, which you can now type in. The password doesn't need to be a full sixteen characters long. Just type in the characters you need, then hit RETURN. Spaces are significant.

This means that a single space is a valid password, and it is different from two spaces! As you type the password, the program shows you how many characters you have entered by replacing the underscores with asterisks. The program will not show your actual password - wandering eyes might then see it when you didn't want them to. If you make a mistake, use the left arrow key to delete the last character typed.

You will soon discover that a single password can be used to encrypt more than one file. Keep in mind that this greatly increases the chances that your files can be broken into. The reason is that when the same password is used an two different files, the same random pattern has been repeated in two places. It may not be obvious that this makes your files vulnerable, but to an experienced analyst, they are. For maximum protection, use a different password for each file. To keep out the curious or to stop a non-programmer, using the same password on several files is not a problem.

Encrypting a File

With the password in place, it is time to use it to encrypt some files. The same password is used until you change it, so you can encrypt a whole group of files using the same password. You might, for example, use one password for all of your correspondence, and another for customer files, but always use the same password for each of these categories. As mentioned, using the same password on several files increases the chances that the files can be broken, but it would take an expert to do it.

Place the disk with the files to encrypt in your disk drive. If the slot and drive numbers displayed at the top of the screen are not correct, change them by typing an S and responding with the proper slot and drive numbers. (For ProDOS, use the same function to select the proper path name.) Now, type an E to start the encryption process. The first few files on the disk will be shown on the display; you can use the U and D keys (or up and down arrows on an Apple //e or Apple //c) to move the menu selection bar up and down the list of files, and S (or the space bar) to select the files you want to encrypt. The files you have selected will have an arrow to the left of the file name. You can deselect the file with the S key as well. Once all of the files you want to encrypt have been selected, use the ESC key to start the encryption. (If you decide not to encrypt any files at all, just use the ESC key right away, with no files selected.)

The program will now step through the menu, encrypting any files that you selected. Once completed, you can leave the program and check to make sure that it did its job.

Decrypting Files

If you can been trying the steps described so far, you now have a disk file which you cannot read or made sense of. By decrypting the file, you can recover the information. Decryption works very much like encryption: you set the proper slot and drive, enter the password, and then select the files to be decrypted. After decryption, the file will look exactly the way it did before you encrypted it.

It is interesting to note that decryption not only works a lot like encryption. but in fact could be used to encrypt a file. This may sound strange, but the point is an important one, so bear with it for a moment. If you try to decrypt a file, and accidentally use the wrong password, the file cannot be decrypted right away with the correct password. instead, you must immediately encrypt the file with the incorrect password, then decrypt it with the correct one. For example, if you encrypt the file with the password MYWORD, then try and decrypt it with the password WRONG, to get your original file back, you must encrypt the file with the password WRONG, then decrypt the result with the password MYWORD.

It is possible to "double lock" a file by encrypting it twice. For example, you could encrypt a file with the password PASS1, then encrypt it again with the password PASS2. This can be useful for cases where you want someone to be able to read a file, but only if two people agree that the file must be read. Giving only one of the passwords to each of the two people insures that both of them must be present to decrypt the file. In the above example, the file should first be decrypted with the password PASS2, then decrypted again with the password PASSI. This method can be extended indefinitely.

Compressing Text Files

Text compression is carried out the same way as encryption and decryption, except that the password is not used, so you don't need to enter one. Although you can try to compress files with any file format, keep in mind that the text compression routines are designed to compress text and nothing else. They might work on other information, but may make it longer instead of shorter!

If you are wondering how a text compression routine could make a file longer, think back to what was said earlier about how the files are compressed. Instead of using eight bits per character, compressed files use a variable number of bits. If your file contains mostly special characters, like unusual punctuation marks, the average number of bits per character could be more than eight. For normal text, this simply won't happen - but if you like experimenting, create a large file of [characters.

Does this mean that only DOS text files (the ones with a T in the catalog) or ProDOS TXT files can be compressed? No, just that the information in the file should be ASCII characters. For example, many text editors write out binary file because DOS can save them faster than a text file, yet the file still consists of ASCII characters.

There are three important limitations that you should be aware of. The first is that the ASCII character set only uses seven bits of each eight bit byte. Some programs set the remaining bit, while others clear it. Crypto can take either type of file as input, and will set or clear the bit when it decompresses the file, but the two representations cannot be mixed. If they are, Crypto will give an error and refuse to compress the file. Every program that we know of uses one format or the other, but not both - still, you should try Crypto on a copy of a file from your editor before trusting it with important files. The next limitation is that the file must end with a $00. Again, all programs that we know of do this. The last limitation is that there must be enough room on the disk for both the expanded and compressed file, since Crypto actually creates a separate file, then deletes the original after the new file is complete.

Reading Compressed Text Files

Once Crypto compresses a text file, it cannot be read by most programs until it is expanded back out to its original form. To do this, simply choose the menu option for expanding text and select the files to be expanded.

Using Compression and Encryption Together

There may come a time when you want to both compress and encrypt a file. The order in which this is done is critical - doing it backwards will not work. First, compress the original ASCII file, then encrypt it. To restore the file, decrypt it first, then expand the result. Compressing an encrypted file will cause an error in Crypto.

Notes to Programmers

This section is for programmers only. Its not that there is anything secret here, just that the information goes beyond that needed to actually use the program. In this section, details will be given about the CRYPTO.OBJ file, which contains several subroutines written in assembly language for maximum speed, Enough information will be given for you to write your own BASIC or assembly language programs which make use of these subroutines.

Parameters

There are six subroutines in Crypto which can be called from a BASIC program. The inputs and outputs for these routines are all contained in page 3, which extends from 768 ($300) to 1023 ($3FF). The inputs are set using POKE statements, and the outputs read using PEEK functions.

The input/output area has the following fields defined. When the entry points to subroutines are described later these areas will be referred to by name.

NAME 768($300) -797($31D)

This field consists of 30 bytes, and contains the DOS 3.3 file name to be worked on. Each of the 30 bytes must contain a character, even if it is blank. DOS normally sets the most significant bit in file name characters. This field is used for both input and output.

Under ProDOS, the name field changes slightly. It consists of 16 rather than 30 bytes, extending from 768 ($300) to 783 ($30F). The first byte is a count byte that tells how many characters are in the file name. The remaining characters are the characters in the file name itself. If, for example, the file name uses only five characters, the last 10 bytes are undefined, and may contain garbage.

LOCKED 797($31D)

This field is only used for the ProDOS version, where the file type doesn't automatically tell if the file is locked. If the ProDOS file is locked, this byte will contain a 1; if the file is not locked, the byte will be 0.

TYPE 798($31E)

This byte contains the file type for the file named in NAME. It is used as an output only; your program does not need to set this field. See the DOS or ProDOS manual for the values that this field can contain.

SIZE 799($31F)

This is the number of sectors (DOS) or blocks (ProDOS) used by the file. Only the low byte is returned, so files larger than 255 sectors or blocks will show up with sizes mod 256.

SLOT 800($320)

Under DOS, this is the slot number to access. The field is not used under ProDOS.

DRIVE 801($321)

Under DOS, this is the drive number to access. The field is not used under ProDOS.

SEED 832($340) - 847($34F)

This is the random number seed to be used by the encryption and decryption subroutines. It is destroyed in the process, so if your program is encrypting or decrypting several files at once, the seed must be set each time a file is encrypted or decrypted. All of the bytes are significant. This means that the value of each byte must be set, not just the bytes from the password.

ERROR 842($350)

If the assembly language routines encounter an error during execution, this field will contain the error number. If no error was found, the field will contain zero. Many error codes can occur due to DOS or ProDOS errors; for those, see the respective operating system reference manual. The errors peculiar to Crypto are:

Number	Meaning
1	File not found - The file in NAME could not be found, so it could not be changed.
2	No data in file - The file was empty, so it could not be processed.
3	File locked - Locked files are not processed.
4	MSB mismatch - To compress a file, all of the characters must either have the high bit set or cleared.
5	Out of Memory - There wasn't enough memory left to open a file, so the command could not be carried out. (ProDOS only.)

Subroutines

With all of the inputs and outputs defined, it is time to look at the subroutines themselves. As mentioned before, there are six of them:

ENCRYPT 32768($8000)

Inputs:	Outputs:
NAME	ERROR
SLOT	
DRIVE	
SEED	

To use this routine, first set the indicated inputs, then do a CALL 32768 from BASIC, or a JSR $8000 from assembly language. If multiple calls are made, be sure and set the SEED input before each call. SLOT and DRIVE are required for DOS only; ProDOS will use the current prefix.

DECRYPT 32771($8003)

Inputs:	Outputs:
NAME	ERROR
SLOT	
DRIVE	
SEED	

To use this routine, first set the indicated inputs, then do a CALL 32771 from BASIC, or a JSR $8003 from assembly language. If multiple calls are made, be sure and set the SEED input before each call. SLOT and DRIVE are required for DOS only; ProDOS will use the current prefix.

COMPRESS 32774($8006)

Inputs:	Outputs:
NAME	ERROR
SLOT	
DRIVE	

After setting the inputs, use a CALL 32774 from BASIC or a JSR $8006 from assembly language to compress the file. If an MSB mismatch occurs, the file will not be changed.

EXPAND 32777($8009)

Inputs:	Outputs:
NAME	ERROR
SLOT	
DRIVE	

After setting the inputs, use a CALL 32777 from BASIC or a JSR $8009 from assembly language to expand the file.

CATINIT 32780($800C)

Inputs:	Outputs:
SLOT	ERROR
DRIVE	

See description of CATALOG, below.

CATALOG 32783($800F)

Inputs:	Outputs:
none	ERROR
	NAME
	TYPE
	SIZE
	LOCKED

In conjunction with CATINIT, this subroutine can be used to get a list of the file names in a directory. It really has nothing to do with encryption or text compression, but is needed to make file names available for menu selection functions.

To use this ability, start by calling CATINIT. If you are working from DOS, set the slot and drive numbers before the call. If ProDOS is being used, set the prefix to the proper directory first. Now call CATALOG. The first file in the directory will be returned with its name in NAME, the file type in TYPE, and the number of blocks or sectors in SIZE. If you are using the ProDOS version, the LOCKED flag will also be set or cleared. In Crypto, these values are saved in an array for future display. The next call to CATALOG will return information about the second file, and so on.