Tiny Cipher 1.5

The installer for Tiny Cipher 1.5 was created using **Installer VISE** from MindVision Software. For more information on Installer VISE, contact:

MindVision Software
7201 North 7th Street
Lincoln
Nebraska 68521-8913
U.S.A.
Voice: (+1) 402-477-3269
Fax: (+1) 402-477-1395
Internet: `mindvision@mindvision.com`
`http://www.mindvision.com`

Introduction

Tiny Cipher is an encryption program. In other words, it takes a file that can be easily read by you (or a third party), and turns it into a new file (of approximately the same size) that is apparently random garbage. Needless to say, it can do the reverse operation to restore the file to its original state. It has been accelerated for Power Macintosh.

Tiny Cipher is a fully-featured Macintosh application, with all the features you would expect in a modern program, such as the ability to drag and drop files and folders onto Tiny Cipher's icon for processing.

Tiny Cipher uses a fast and secure cryptographic algorithm, the **Tiny Encryption Algorithm.** This provides near-military grade security coupled with extremely fast processing. Tiny Cipher is about three times faster than good commercial implementations of the Data Encryption Standard while providing higher security.

Short Cryptography Tutorial.

I. Why do we need cryptography?

Encryption becomes more and more necessary every day. Sensitive information of all kinds is now stored on computers, and there is often little defence against its being read by unauthorised persons. There are several ways to guard against this. The first, and most obvious, is to deny unauthorised persons physical access to the sensitive data. In other words, the computer and hard disk is in a locked room to which you have the only key. This is called **physical security**. It has a number of serious drawbacks, especially in an environment like an office where several people might require access to the same machine, but are not allowed access to all the files. The second method is common in multi–user operating systems like UNIX: everybody has a **password**, and is only allowed access to his or her own files and a restricted set of other files. This is fine for most purposes, but it is vulnerable to attack if the security of the system is not absolutely first–rate. UNIX is a notoriously insecure system, and with the growth of internetworking, systems are susceptible to attack from geographically distant quarters. The third method is to hide the file in such a way as to make it look like it is something else, or to make it impossible to find. This approach is called **steganography**, from the Greek for concealed writing, and if used carefully can be extremely effective. Unfortunately, most situations do not lend themselves well to this method. So we are left with the fourth method, **cryptography**, which is Greek for secret writing. A system which implements a cryptographic algorithm (or **cipher**) is known as a **cryptosystem**. Tiny Cipher is one such system. The science (and art) of breaking (or attacking) cryptographic algorithms and systems is called **cryptanalysis**. Both cryptography and cryptanalysis come under the banner of **cryptology**.

The Libertarian Bit

Governments are making increasingly unfriendly noises towards computer users. If you don't want the government to read your email, then you'd better encrypt it. You don't have to be doing something illegal to benefit from encryption. The more nefarious of the Western intelligence agencies (for example the DGSE in France) do not think twice about reading their citizens' private communications (hell, the DGSE blew up an entire ship — Greenpeace's Rainbow Warrior in New Zealand* — why would they worry about the legality

* This was a bit over the top, even for those nefarious Frogs. They killed a perfectly innocent photographer and sank a foreign-owned ship in New Zealand waters. Precisely why the RNZN gunboats didn't sail on the next tide is a mystery to me. The diplomatic response was along the lines of 'Tut Tut, please don't do it

of things like reading your email?). Strong encryption is forbidden in France, which is why I encourage any French readers to use my software as much as possible. Even governments in countries where intercepts on communications channels such as phonetaps have to be authorised at a high level, like the US and the UK, are not above snooping on people for no good reason. The only way to get them to stop it is to make it not worth their while. If every email that anyone sent was routinely encrypted, then there would be no point in trying to snoop. The effort required would simply be impossible to justify. And what if your computer was stolen? Do you want your private stuff readable by the bloke in the pub that buys your computer off the thief? Since a file encrypted with a strong cipher is indistinguishable from a stream of random numbers, there is nothing for the authorities (or anyone else) to get their hands round. If all the traffic on the world's networks looked like line noise, then the snoopers would be unable to do their job. All cats look grey in the dark. Strong cryptography is not a "pornographer's charter" or whatever. If it were, I wouldn't want to be associated with it. I would dispute that there has ever been a crime committed where, if strong cryptography had been outlawed, the authorities would have prevented it (the crooks will just use strong crypto anyway). On the other hand, Phil Zimmerman's PGP secure email program is allowing pro-democracy campaigners in Burma to do their stuff without getting shot in the back of the neck and dumped in a shallow grave. **Cryptography is a libertarian issue**. Just like guns, if you outlaw cryptography, then only outlaws will use it. By the way, for you US readers, there is a **whole boatload** of First Amendment stuff (and Fifth, too) concerning cryptography that just hasn't been properly aired yet. Bearing in mind that DTR classes cryptography as a "munition", there's probably a constitutional challenge under the Sixth Amendment as well!

The UK government is currently working on proposals implementing some form of key escrow, whereby you can use strong crypto as long as you hand over your key to a 'trusted third party' (TTP). When the government wants to break one of your communications, it applies for a court order, then it goes to the TTP which releases the key. Since governments in the West are tirelessly working for the good of their citizens, there's no civil liberty or privacy issues at stake here. And the moon *is* made of green cheese. You'll see the Pope and her husband at their eldest son's Bar Mitzvah before you get me believing that. What the technical illiterates in our beloved People's Revolutionary Jamahiriya (prop. A. Blair) don't realise is that if you super-encrypt your messages with something like Tiny Cipher, then schemes like key escrow fail dismally. They get a court order, and decrypt your email, and get two pages of line noise. But we must stop the Evil Porn Barons™ and Forces of International Terrorism®. In the words of Sir Humphrey Appleby in 'Yes, Minister', "Something must be done. This is something. Therefore we must do it." The logical flaw in this particular syllogism is not hard to find. The other thing that is conveniently brushed aside is that once they have your key, they can read every other message encrypted with that key, without the bothersome effort of getting another court order. The UK courts recently decided after being lent on by the Home Secretary Jack Straw (who makes his predecessor Michael Howard look like a pinko), that emails are not subject to the same privacy issues as surface mail, and the cops can snoop on them to their heart's content. Get a copy of PGP, super-encrypt your emails with Tiny Cipher, and laugh.

It's probably safest to assume that your government is an intrinsically malevolent institution, and work from there. Your government probably isn't intrinsically malevolent, but when in a film did you see one of the

---

again'. This 'incident' still makes me angry today. I imagine the photographer's widow isn't all that happy about it, either. Safeguard your files as a way of sticking one/two (depending on where you come from) fingers up at the evil sods who think that you don't matter.

characters survive to the last reel if they assumed that the hideous shape-shifting mutants weren't out to get them? Precisely. Assume your government is run by George Romero, and encrypt your files. This will cause less grief in the long run. At least when your local shopping mall is being overrun by flesh eating zombies your files will be secure.

Oh dear. How sad. I sound like a nasty right-wing survivalist weirdo. Nothing could be further from the truth.

OK. Rant over.

II. What a good cryptographic algorithm should do. (Caution: contains Technical Words™)

A good cryptosystem should be able to take a file, assumed man-readable* (this is called the **plaintext**), and convert it into a form that cannot be read by anyone else (this is called the **ciphertext**), under the control of a piece of secret information, called the **key**. Moreover, it should be nearly impossible to recreate the plaintext from the ciphertext without knowledge of the key. The crucial point about a good cryptographic algorithm is that this should hold true even if full details of the algorithm are known to a potential attacker . **This is important!**. An algorithm that relies solely on knowledge of its workings for its resistance to attack represents a trivial challenge to a competent attacker. A cryptographically secure algorithm may also be restricted, but it is always assumed when designing an algorithm that eventually an attacker will learn (or deduce) the inner workings of the system. **All the security of the algorithm lies in the key**. You can assume that a cryptanalyst can perform at least a **chosen–plaintext attack**, where the cryptanalyst can run plaintexts of his choice through the algorithm, in addition to having several known plaintext/ciphertext pairs. I've gone as far as including source code for TEA in this documentation. I'm confident TEA is a strong (actually, *very* strong) cipher.

The difficulty an attacker has in breaking a particular algorithm must be as high as possible. The ideal situation is for the cipher to be completely unbreakable. Sadly, there is only one truly unbreakable (or **unconditionally secure**) cipher, the **one–time pad**. This is nothing more than a list of completely random data, each of which is combined exactly once with the corresponding datum in the plaintext. If the pad is generated using a truly random process, then it is uncrackable. Since all pads are equally likely, the decryption of a given ciphertext can be any message. There is no way of telling the true message from all the others. However, the need for the pad to be the same length as the plaintext, and for it to be discarded after being used just one time, makes this type of system impractical for all but a few specialised uses (Soviet agents in the West used one-time pads based on tables of economic data, and some of 'em got fried because the glorious people's democracy was so lame they couldn't issue enough one-time pads and the agents had to re-use them, which defeats the whole point). If, however, a system can be devised that generates lists of data in a systematic way, such that the list is indistinguishable from truly random data, then that sequence will make a cryptographically secure sequence. This is known as a **cryptographically secure pseudo-random number generator** (CSPRNG), which is what cryptographic algorithms aspire to be. To be cryptographically secure, the output of the algorithm must not only be statistically random (i.e. every symbol is equally likely) but it must be infeasible to ascertain a priori what the next symbol will be given complete knowledge of the algorithm and all previous output symbols. It's also notable that from a mathematical standpoint, a CSPRNG makes a beautiful random number routine for things like Monte Carlo simulations. The drawback is

* insert favourite gender inclusive phrase here

that they're typically much slower than inferior techniques like linear congruential generators. But I digress…

'Infeasible', in the cryptanalytic sense, means that with the resources at the attacker's disposal, it is impossible for him to break the cipher in a realistic time. The goal of any designer of a cryptographic algorithm is to make the most efficient search a 'brute force' attack. In other words, the fastest way to break the cipher is to try every possible key. Until cryptologists come up with a better method for a new algorithm, the attacker is left with brute force. Clearly, then, the length of the key is all–important. The number of different keys is 2^n, where n is the length in bits. A 40 bit key has only about 1 trillion combinations. It has been estimated that a hardware device capable of breaking this key within a week would cost well under £5,000 at 1997 prices. The pathetic 40 bit keys allowed for export under US Defense Trade Regulations (DTR) have been shown (Dateline: January 1997) to be crackable in 3.5 hours using a fairly fast workstation. To break a 64–bit key in a week in 1995 would have required a machine costing over £500 million. However in 2005, this will have dropped to less than £20 million, well within the intelligence budget of an advanced industrial nation. For even less than this, a machine to break a 56 bit key (such as DES uses) in an hour could be built in 2005. The Electronic Frontier Foundation (http://www.eff.org) have built and demonstrated a hardware device which can break DES in three days, for under $250,000. The crucial factor is Moore's law, which states that the computing power available for a given amount of money doubles every eighteen months. A doubling of computing power allows a key one bit longer than before to be cracked in a given time. This means that even a 64 bit key will become useless in 25 years' time. A 128 bit key, on the other hand, which has ~3.4 10^{38} combinations, requires more computational power than is ever likely to be available. A machine costing £1 trillion (which is around the same size as the UK's entire annual GDP) would take nearly 6 trillion years to guarantee it would find the key. Even a machine which could perform a trillion trillion tests a second would take five million years to have an odds-on chance of cracking a 128 bit key. Current (1998) top of the range supercomputers are about fourteen orders of magnitude slower than this. If you are using Tiny Cipher to encrypt your accounts then it is unlikely that your government will spend its entire revenue only to break the encryption some time around 3 trillion AD.

III. The Tiny Encryption Algorithm.

There are literally hundreds of encryption algorithms. Cryptography dates back to antiquity. Most ancient ciphers are easy to break by today's standards. Modern cryptography dates from about the middle of the 19th Century, but it was not until after the First World War that cryptology was first seriously studied in a scientific manner. Almost all this research was done by the military; civilians had little or no access to communications security. This situation persisted until about 1970. Suddenly there was an explosion of interest in cryptography among the open academic community. The field is now extraordinarily rich, with dozens of new papers appearing each month in the literature.

To be an adequate cryptologist, you need to be pretty good at mathematics. To be a good cryptologist, you need to be a superb mathematician. All modern, open ciphers are released to the academic community with a detailed and erudite mathematical analysis of their properties. Many of these algorithms are extremely complex (although complexity alone is not enough to guarantee security). The problem with a complex algorithm is that it will have a slow execution speed on a small computer. What is needed is an efficient algorithm that is also secure.

David Wheeler and Roger Needham, at the Cambridge Computer Laboratory, devised one of the simplest and fastest algorithms to date, called the **Tiny Encryption Algorithm** (TEA). In its C–language form, the encryption and decryption routines are no more than six or seven lines long. The algorithm consists of a simple combination of shifts and exclusive–ORs, repeatedly applied to a 64-bit plaintext block under the influence of a 128-bit key. One iteration of the algorithm is called a **round**. The number of rounds can be variable, although eight suffices for most applications and 32 is overkill. Raw throughput on an obsolete machine (for example a Mac IIsi, with a 20 MHz 68030, like the machine on which I created the earlier versions of Tiny Cipher) is more than 1.8 million bits per second for TEA with eight rounds. When all the adornments of a graphical user interface are grafted on, plus reading and writing to disk, then this figure drops to about 40% of this value. That still works out to about 83K a second, or about 12 seconds per Megabyte. That is much faster than most implementations of more complex algorithms, such as the Data Encryption Standard (DES). On a faster machine, performance will of course be better than this. On a very powerful machine, like a Power Macintosh, then the fundamental limit is how fast the machine can transfer data to and from the hard disk drive. On my 8600/250 I obtain raw encryption speeds of 64 million bits per second for twenty-four round new-variant TEA. Even with the fast internal SCSI 3 bus on this machine, Tiny Cipher is limited to about 2.5 Megabytes per second (this is still 30 times faster than the IIsi!). The SCSI bus simply cannot keep up with the torrent of data that is being fed through it. If you are reading and writing a RAM disk then the speed of Tiny Cipher is awesome to behold (10 Mbytes per second on a G3/266!). Tiny Cipher uses 24 rounds for added security (16 for v1.4 files), but even then it is feeding the disk faster than it can keep up. To put it in perspective, TEA is quite capable of encrypting a broadcast-standard video stream in real time.

So, is the TEA cryptographically secure? So far, it appears to be. The algorithm has been in the open literature for over three years now, and there have been no reports of any successful cryptanalyses. One of the key criteria of a good algorithm is that it should have strong diffusion properties. Information in the plaintext should be rapidly spread out into the ciphertext. In other words, if **one** of the bits in a plaintext block changes, approximately **half** the bits in the corresponding ciphertext block should change, apparently at random . The TEA satisfies this criterion after only six rounds, and then maintains it indefinitely.

A graduate student at Berkeley in California, David Wagner, made some observations about two very minor weaknesses in the original version of TEA. One was a *related-key* attack, in which pairs of keys with a given relation between them are used to encrypt data, and from this relationships between the plaintexts are deduced. To mount a related-key attack against TEA, the cryptanalyst needs to assemble 2^{32} texts. This is many Gigabytes of storage. He then needs to process these texts, looking for the relations. But this is only the case when considering a *chosen-key chosen-plaintext* attack. If the attacker does not have the liberty of choosing the plaintext (and he won't, in practice), then the number of texts needed goes up astronomically (i.e. beyond any feasible attacker's resources). This weakness is of *no real importance to the security of Tiny Cipher*.

A second weakness was in the *key schedule*, that is to say, the way in which the key material is mixed into the algorithm along with the plaintext to form the ciphertext. David Wagner noted that the key schedule of original TEA (which I'll call old-style TEA from here on) meant that the effective key length was reduced from 128 to 126 bits. This is a negligible weakness, as 126 bits is still ample to secure your files for all time. However, Roger Needham and David Wheeler adjusted the algorithm to repair the two weaknesses, and with version 1.5 of Tiny Cipher, I followed suit. Version 1.5 can still read (and write) version 1.4 files.

Of course the reverse is not true: if you want to decrypt a file that has been saved in v1.5 format, then you'll need Tiny Cipher 1.5.

Another very powerful cryptanalytic technique is that of differential cryptanalysis . The precise details of this technique are extremely mathematical and require a good knowledge of linear algebra and number theory. In essence, the attack is a statistical one; pairs of plaintexts and ciphertexts are examined, and if they satisfy certain criteria then some knowledge of the key can be gleaned. TEA is very resistant to differential cryptanalysis. Even if it were not, gathering the statistics to launch an attack is a daunting proposition. The number of plaintext/ciphertext pairs needed to perform this attack is in the trillions.

Tiny Cipher should be capable of encrypting files to a level of security that is safe against even the resources of national governments. If you want to use it to send secure email, for example, then it should be proof against anything GCHQ, the NSA or the DGSE can throw at it.

To sum up, Tiny Cipher is capable of encrypting files to a degree of security that equals or exceeds that of most (probably all) commercial Macintosh encryption products. It is totally, utterly and unequivocally free, and it does more things, better, than most standalone cipher packages that cost you many clams. The algorithm is open, I have nothing to hide, and if you ask nicely how I've done something, I'll give you (yes, give you) the source code. It may turn out that the TEA has a hideous flaw in it that makes it totally insecure and that Tiny Cipher is utterly useless. But I doubt it. If you follow a few rules of thumb, then you can knock out a pretty secure cipher with little difficulty. It's a little appreciated fact that cryptography outstripped cryptanalysis a while back. A successful attack on a cryptosystem (like the Netscape hack a while ago, and the recent attack on the Secure Sockets Layer) will almost certainly be focused on the key generation/ management side of the system rather than the algorithm itself. No-one bothers to attack the underlying cryptographic algorithm anymore, because they know it's secure. In other words: keep your password safe, and only you can read your files, full stop.

Finally, one word of warning: Tiny Cipher is mainly intended for use as a standalone system for encrypting single files or folders for one user. It's not really intended for things like secure email. However, having said that, if you are one of the unfortunate individuals using an export-strength version of PGP with the risible 40 bit key, then Tiny Cipher can be used to super-encrypt your messages. Quite how you will perform the key distribution in this situation is another matter…

Using Tiny Cipher

Tiny Cipher is very easy to use. The commands Encrypt… and Decrypt… in the File menu do exactly what they say. You will be presented with a directory dialogue box allowing you to choose a file or folder for encryption or decryption. Then you will be presented with a dialogue box asking you for a password. This is the sequence of characters that will be converted into the key for encrypting the plaintext. Obviously, all the security of the system hinges on how difficult it is for an attacker to regenerate the key, knowing the plaintext. If he can guess the key, or find it out in some way other than cryptanalytically attacking the cipher, then you might as well not bother with encryption at all. Although TEA uses a 128–bit key, and is therefore potentially very strong, by choosing weak passwords you reduce the number of keys that can be generated, and hence the number of different ciphertexts your plaintext can be encrypted to. The first point, and this is absolutely crucial, is to choose a password that is hard to guess. A so-called 'dictionary attack' uses a huge list of common names, words, nicknames, acronyms, mnemonics etc. as a password generator. It uses these as an attack on the cryptosystem. Up to 25% of the passwords on most multi-user systems can be cracked this way. Do not use your name, or your mother's name, or your favourite football team's name, or place names, cartoon characters, biblical quotes, Shakespearean quotes, or permutations of any of these. And don't for God's sake, whatever else you might do, ever, ever write your password down.

It's important to make your password a reasonable length. If your password is six characters long, and you only use the 26 lowercase letters, then the number of possible keys is only 309 million, which is a minute fraction of the total number of possibilities that a 128–bit key has. If, on the other hand, you choose a random 16 character password with each character allowed to come from the 95 printable ASCII characters then you can reach about one ten-millionth of the total 128–bit keys, which is pretty good. However, a password like £1µ5eÆB‡.WÇΔt=¬¤ is quite hard to remember. The best choice is a compromise. Choose a password that is easy to remember, but hard to guess. A good technique is to choose two or more unrelated but memorable words (remembering to add some capitalisation) and separate them with a punctuation mark. For example, proPerty$snEEze , or angLe!kidnEy. Another technique is to choose a very long pass phrase, such as a quote from a poem or book, and use that. Tiny Cipher is quite happy with passwords up to 128 characters long. For example, "In the centre of the city a woman stood between two mirrors, watching herself reflected all the way to infinity." While this has the advantage that it is easy to remember, and long, it can be a bit tiresome to type in. And since by default Tiny Cipher will not echo your typing to the screen, a spelling mistake is possible (and disastrous). The choice is yours. To put it in perspective, an exhaustive search for a *random* eight-character password using all 95 printable ASCII characters would take 210 years if the search engine could make one million attempts per second. For a *random* eight-character password using only lowercase letters it would take 2.4 days. A ten character password using the 62 characters a–z, A–Z and 0–9 *at random* would require 26 600 years. The reason I stress the word random is that ordinary English (or any other natural language) is far from random. There are no words in English that start with the letters 'chn', for example. If English words were allowed to be any combination of letters, then the whole English vocabulary could be covered by words of 4 letters or fewer (this allows 475 254 combinations). That they are not means that English has *redundancy*, which makes guessing a password a lot easier. To get the randomness using English words, you need longer passwords. Tiny Cipher by default imposes a minimum of 9 characters in the password (you can set this limit to one character), but after that what you type is up to you.

One other word of warning. If you forget your password, or mistype it when you are encrypting, then to recover your file you will essentially have to go through the same procedures a cryptanalyst would if he wanted to decrypt that file. A simple typing error is comparatively easy to rectify, since the number of combinations of different passwords is small, assuming you are not completely ham-handed. Forgetting your password is another matter. You might wish to try deep hypnosis before starting a degree in algebraic number theory. At the request of users, the ability to echo the password in plain text to the screen has been added in version 1.2. This reduces the risk of mistyping the password, but obviously gives less security against people viewing the password. 'Ware kibitzers!

Teacrypt Compatibility

From version 1.4, Tiny Cipher is capable of reading and writing files in a format that is compatible with the other encryption utility using TEA that I have written. This is teacrypt, which is intended as a drop-in replacement for the insecure crypt(1) that is found on Unix systems. There are public domain cracking engines available which can decrypt a file enciphered with crypt fairly easily. I wrote teacrypt to give people a more secure option for encrypting their files. Teacrypt is much simpler than Tiny Cipher – in use it functions much like zip. There are three versions available for download from my ftp site, one for HP-UX 9.03 or later, one for Solaris 2.5, and one for Solaris 2.6. To encrypt a file in teacrypt mode, hold down the option key. You will see that the menu item has changed to TEA Encrypt… .

At this point, the differences between the Mac file system and others becomes apparent. Under Unix (and PC) file systems there is no concept of the split fork (data and resource) structure of Mac files. The only kind of files that can be encrypted in teacrypt mode are those with a zero-length resource fork. Conversely, on decryption, teacrypt files produce data fork-only files. The other difficulty arises from the problem of associating files with their creator applications. There is no equivalent of file type and creator under DOS or Unix; the former uses the three character suffix to denote the type of the file, the latter has no file/application association whatever. To get round this, I do a rather sneaky thing. Tiny Cipher looks in the Preferences folder for the PC Exchange Preferences file, and uses that to map DOS suffices to file types. I recommend adding type 'TeaC', with Tiny Cipher as the creator, for files with the suffix .TEA in the PC Exchange Control Panel, if you are going to be regularly transporting encrypted files between Mac and Unix.

Tiny Cipher does intelligent decryption: if it sees a file of type 'Cryp' or ‘TC15’ with creator 'TEA*', or with the suffix .crypt, then it assumes it is a normal Tiny Cipher file. If it sees a file of type 'TeaC' and creator 'TEA*', or with suffix .TEA, then it assumes it is a teacrypt file.

Teacrypt is limited to eight character passwords. Passwords larger than this will be silently truncated. This is a limitation of the Unix getpass() function; I might allow longer passwords in later versions of teacrypt and if I do, Tiny Cipher will be updated to reflect this.

New TEA vs. v1.4 Format Files

The new variant of TEA is used to encrypt files by default. If you want to encrypt files using old-style TEA (perhaps to send to a friend who is using v1.4 or earlier of Tiny Cipher), then hold down the Control key. You will see that the Encrypt… item has changed to v1.4 Encrypt… . Operation is in all other ways identical to Tiny Cipher 1.4 and earlier. Decryption of v1.4 format files is handled transparently.

Secure File Deletion

Tiny Cipher can also securely delete files. One of the most powerful resources an attacker has in breaking an encrypted message is to have a plaintext/ciphertext pair for a different message encrypted with the same key as the one he is trying to break. Simply throwing the file in the Wastebasket is insecure, since this does not immediately erase the data from the hard disk but simply changes some information in the disk catalogue to indicate the space the file occupied is now ready for re-use. The file can exist (maybe in fragmentary form) for months. To be securely erased, the file must be overwritten before being deleted. Tiny Cipher does this by overwriting the file several times. The number of times is set by you in the ‘Preferences…’ dialogue box. You can choose between three, five and seven times. The first two overwrites store a stream of alternating ones and zeroes to disk (the second sequence has zeroes where the first has ones). Then one, three or five random number sequences are written to disk. Finally the file is deleted in the normal way.

This function can be performed by using the ‘Delete…’ command in the ‘File’ menu. You will be presented with a dialogue box asking you to choose the file to erase. You will always then be asked to confirm this choice. Alternatively, you can allow Tiny Cipher to automatically delete files after encryption and decryption. Choose the ‘Auto-delete source files’ option in the preferences dialogue box. You can choose here whether Tiny Cipher should warn you that it is about to auto-delete a file.

Note that from version 1.3, I have included the ability to delete entire folders full of files. Allowing folder deletion potentially allows enormous mayhem. Norton Recover will not help — the information in your files has been lost beyond all hope of recovery. If you are the CIA, then you may be able to get some fragmentary data by tuning the head to scan the edge of the track instead of the middle, turning up the maximum likelihood detector to full wick and crossing your fingers (even this is likely to fail with the high security deletion option, since various results indicate that residual magnetised regions only persist through about five overwrites). Otherwise, the data is **GONE**. I cannot stress this too highly: inadvertently delete a file using the secure erase and it ceases to exist. You will do this at most once, and then you will learn the secret of regular backups: “it's not whether you’re paranoid, it’s whether you’re paranoid **enough**.” However, if you feel that your life would be incomplete without folder erasure, then check the ‘allow folder deletion’ checkbox in the preferences. But don’t say I didn’t warn you. You can kill a disk with your life’s work on it in a heartbeat.

If Tiny Cipher is doing a long encryption or decryption (or secure deletion), a progress box will appear, like the one you see when you perform a copy operation in the Finder. Tiny Cipher can then be put in the background to work on the file while you do something else. You can then choose the method that Tiny Cipher should use to inform you that the operation has completed. You can choose between no notification, flashing an icon in the application, and posting a note on the screen. You won’t often need to do this. Tiny Cipher is fast enough for even big files to only take a few seconds.

When you enter your password, it will normally be echoed to the screen as a row of bullets (•). If you do not want someone looking over your shoulder to know how long your password is, then you can alter this behaviour. Uncheck the ‘Echo password’ option in the preferences dialogue. As from version 1.2, the password can be echoed as plain text. Use this option with care if you are working in an environment where someone could be looking over your shoulder.

As from version 1.3, I have included filename scrambling. Normally when Tiny Cipher encrypts a file, the name of the encrypted file is simply that of the source file with a suffix appended to it (by default, ‘.crypt’). The suffix can be changed in the preferences, up to a maximum of seven characters. If filename scrambling is enabled in the preferences, then the new file is given a name of the form AAAAaaaa0000, e.g. AAAAaaad7834. This prevents an attacker from easily associating plaintext files with their corresponding ciphertexts. Note that using the secure deletion facility is a much better way of preventing this attack.

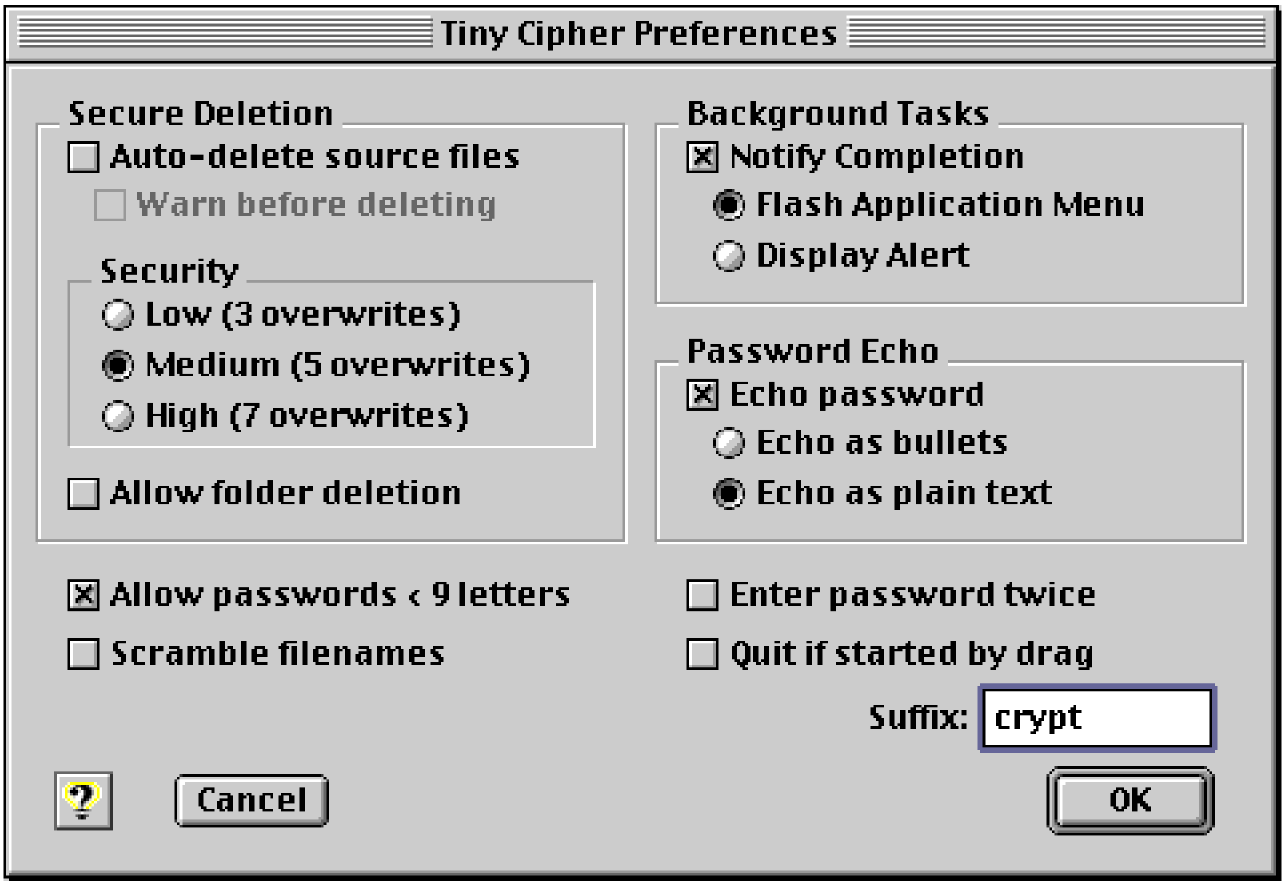
A Strange and Mysterious Phenomenon

Note that there is a subtle and strange ‘feature’ that you may see. Some packages (notably BBEdit) have the bizarre and totally unsanctioned behaviour of only opening files when they are being written to disk. An application is meant to maintain exclusive access to any files it is using (this can be at the level of the whole file, or merely at the level of portions of the file). Files can be opened ’single write/multiple read’ so that while only one user can alter the data in a file, several people can read these data. However this only pertains in shared environments; in general an application opens a file for reading and/or writing for the whole time that file is required. If it does not, and closes the file, then the operating system has absolutely no way of knowing and another application can alter (or even delete) the file without any problems. This is not a good idea. To see what I mean, you can do the following experiment if you have BBEdit 3.5. Create a folder, and a file in that folder with BBEdit. Make sure the file is saved in BBEdit, then go out to the Finder and drag the folder and contents to the Wastebasket, and empty it. Then go back into BBEdit, alter the file, and try to save it. BBEdit will come back with an error saying the directory couldn't be found! If BBEdit had behaved itself and done things the way it was meant to, you couldn't have done this.

The upshot of this is that if you have a file open in an application that behaves in this silly way, then you can securely delete that file without Tiny Cipher knowing. This is not a bug in Tiny Cipher. It's a bug in the other application. Tiny Cipher does check to see if files are open before trying to overwrite them, but it can't (and couldn't be expected) to know if they are still active somewhere else. It's bizarre that BBEdit behaves this way, as it's so well-written in other respects.

Preferences

The preferences dialogue box is shown below. A description of each of the items follows.


Tiny Cipher Preferences

Secure Deletion
- ☐ Auto-delete source files
  - ☐ Warn before deleting
- Security
  - ○ Low (3 overwrites)
  - ● Medium (5 overwrites)
  - ○ High (7 overwrites)
- ☐ Allow folder deletion

Background Tasks
- ☒ Notify Completion
  - ● Flash Application Menu
  - ○ Display Alert

Password Echo
- ☒ Echo password
  - ○ Echo as bullets
  - ● Echo as plain text

☒ Allow passwords < 9 letters
☐ Scramble filenames
☐ Enter password twice
☐ Quit if started by drag

Suffix: crypt

Secure Deletion. The options in this group control the way Tiny Cipher automatically deletes files and folders in a secure fashion. To automatically erase files after encryption or decryption, check the ‘Auto-delete source files’ box. To have Tiny Cipher ask for confirmation of secure deletion, check the ‘Warn before deleting’ box. This option is only available if the ‘Auto-delete source files’ box is checked.

The radio buttons in the ‘Security’ sub-group control the number of times the source file is overwritten. Low, medium and high security overwrite the file with three, five or seven different sequences consisting of two sequences of alternating ones and zeros with the second being the complement of the first, followed by one, three or five random number sequences respectively.

To allow Tiny Cipher to securely delete folders and all their contents (including sub-folders), check the ‘Allow folder deletion’ box. Use this option with care!

Background Tasks. The options in this group control how Tiny Cipher notifies you when it has completed a task after having been switched into the background. To receive an acknowledgement that processing has finished, check the ‘Notify Completion’ box. To receive this notification in the form of a flashing icon in the Application menu, select the ‘Flash Application Menu’ radio button. To receive the notification by means of a notification alert box, select the ‘Display Alert’ radio button. The radio buttons are only available if the ‘Notify Completion’ box is checked.

Password Echo. The options in this group control the way in which the password is displayed in the password entry dialogue box. To have the password echoed to the screen while typing, check the ‘Echo password’ box. If this box is unchecked, then no visible indication of typing will be seen. To have the password echoed to the screen as bullets (•), select the ‘Echo as bullets’ radio button. To have the password echoed to the screen as text, exactly as it is typed, select the ‘Echo as plain text’ radio button. The radio buttons are only available if the ‘Echo password’ box is checked.

Password Length. To allow the choice of passwords that are less than nine characters in length, check the ’Allow passwords < 9 letters’ box. Note the maximum password length is 128 characters whether or not this box is checked.

Filename Scrambling. To have Tiny Cipher generate an unintelligible filename for an encrypted file, check the ‘Scramble filenames’ box. If this box is unchecked, then encrypted files will by default have a name which is the same as the source file with the suffix ‘.crypt’. This suffix can be changed in the ‘Suffix’ edit box. Any suffix up to eight characters (full stops / colons excepted) can be entered here.

Password Entry. To make Tiny Cipher require you to enter the password twice, check the ‘Enter Password Twice’ box. This option is a useful safeguard against mistyping a password, especially when the password is not being echoed as plain text.

Drag and Drop Behaviour. To make Tiny Cipher quit after processing a file if has been launched by dragging an icon or icons from the Finder onto its icon, check the ‘Quit if started by drag’ box. This behaviour mimics applications like Stuffit Expander, which are primarily intended as ‘droplets’, i.e. applications that are meant to be triggered by dragging files onto them. If this box is unchecked, then Tiny Cipher will remain running after having been started by dragging onto its icon. In either case, if it is already running, dragging an item onto its icon will not cause it to quit.

Balloon Help. Clicking the question mark lightbulb icon in the bottom left of the dialogue box will toggle Balloon Help on and off.

To confirm your choices and update the preferences file, click the ‘OK’ button. To exit the preferences dialogue box and leave the preferences file unchanged, click the ‘Cancel’ button. To return the preferences to their defaults, locate the file ‘Tiny Cipher Preferences’ in the Preferences folder in the System Folder and drag it out of the System Folder. The next time you alter the preferences from their default settings, a new version of this file will be created in the Preferences folder.

Cosmetic Enhancements

With the release of MacOS 8, Apple have introduced a number of new look-and-feel technologies. Tiny Cipher supports two of the latest: the Appearance Manager and Navigation Services.

The Appearance Manager is a complete upgrade to the user interface (which has remained unaltered in many respects since 1984, and in virtually all respects since System 7 came out at the start of the nineties). The MacOS 8.5 user interface has the ability to support ‘themes’ — complete suites of windows, menus, buttons and controls all with their own consistent look. Appearance Manager-compliant applications will switch seamlessly between these themes on the fly. The only theme released with MacOS 8.5 is the ‘platinum’ look, which looks just like the 8.1 style. Additional themes will be released in due course, perhaps with MacOS 8.6. I’m sure there will be third party themes out there as well.

Navigation Services is an overhaul of the Standard File Package. The old-style directory dialogue boxes have been replaced with much more powerful modeless dialogues (i.e. you can switch to another application while they are open) which can be moved and resized. In addition, they allow you to select multiple files. To take advantage of Navigation Services, see the System Requirements below.

System Requirements

Tiny Cipher requires System 7.0 or later. A lot of its more advanced features need 7.5.5 or later. It is fully compatible with the latest system versions up to 8.5. It will run on any machine with 128K ROMs or better. In practice, this means everything from a Mac Plus to a 400 MHz G3 machine. The 68K version needs about a Megabyte of RAM to be truly happy, although it will run in 600K. There's no advantage in giving it more than about 1500K of RAM. The Power Mac version needs 2.5 Mb, but giving it more will help speed things up. After about 5 Mb the speed increases tail off. The reason for the disparity is the fact that the Power Mac versions can encrypt so much faster than talking to the disk. A bigger RAM entitlement means bigger chunks of the file can be read at a time, which speeds things up a lot.

To get the ‘platinum’ look and feel you will need Mac OS 8 or later. That is the first operating system version to truly support the new Appearance Manager. Themes support requires 8.5 or later. To get the new Navigation Services support, you will also need Mac OS 8 or later (Mac OS 8.1 or later recommended), and for system versions earlier than 8.5, the ‘Navigation’ shared library (this library comes as standard with

8.5). The Navigation library is available to download from Apple's developer web site as a software development kit, or from my ftp server, and is highly recommended if you are running 8.1 and have no plans to upgrade to 8.5. There is no support for either Appearance Manager or Navigation services in the 680x0 version of Tiny Cipher (Mac OS 8 requires a Power Macintosh or 68040, and I don’t have a 68040, so I couldn’t test it. MacOS 8.5 is PowerMac only).

To support the online manual and clickable hotspot in the About box, using MacOS 8.1 or earlier, you will need the Internet Config 1.4 preferences management system, and a reasonably up-to-date web browser (version 3 or later of either Microsoft Internet Explorer or Netscape Navigator/Communicator will be fine). The functionality of Internet Config has been rolled into 8.5 as the Internet Control Panel. It is essential that you configure your Web browser to be the “http” and “file” helper in the “Helpers” section of Internet Config/Internet Control Panel. Internet Config is highly recommended, even if you do not have an Internet connection, as it handles many useful functions (like mapping filetypes to Macintosh applications) that are not Internet specific. It’s also completely free. The complete package is available from:

`ftp://ftp.quinn.echidna.id.au/Others/Quinn/Config/`

and

`ftp://ftp.share.com/internet-configuration/`

These are the Internet Config home sites; it is also available in several places round the Web, such as umich-mac and sumex (and their mirrors) .

Internet Config and the Navigation Services library are distributed as part of the ‘full’ Tiny Cipher package.

Distribution Details and Copyright.

Tiny Cipher is copyright © David A. G. Gillies 1995-1998. All rights reserved.

Tiny Cipher is freeware. That means you can give it to your friends, install it on your machine at work, put it on a disk of free software, whatever. I encourage you to spread it as widely and freely as possible, especially within France and the United States of America (DTR? Phooey). Tiny Cipher is **NOT PUBLIC DOMAIN**. That means you cannot charge for it, and the copyright remains vested in me. You cannot pass it off as your own, or modify it and pass it off as your own. If I divulge the source code, or portions thereof, or divulge techniques, structures and algorithms that govern the operation of Tiny Cipher to you, then the copyright is still vested in me.

I wrote Tiny Cipher to give highly secure, fast cryptography to Macintosh users. That's why it's free. Quite a lot of work (actually a fairly humongous amount) has gone into this task to make Tiny Cipher a fully-functional Macintosh application. I would appreciate your comments (and any bug reports) by email to

`david@vader.eeng.brad.ac.uk`

If you're reading this file and can't find a copy of Tiny Cipher then you can obtain it by anonymous ftp from my ftp server at `vader.eeng.brad.ac.uk`. The download page is available on my web site at

`http://www.vader.brad.ac.uk/tea/tcdownload.shtml`

There are several download options, including a combined installer which will install the correct version for your architecture (it also has a custom install feature).

Check out the TEA/Tiny Cipher web pages at

`http://www.vader.brad.ac.uk/tea/tea.shtml`

Other stunning bits of software I have written

You might want to check out **QuickLaunch**, a replacement for that clunky Launcher thing Apple gave us with System 7.5. **QuickLaunch** allows you to define sets of documents and applications which can then be launched as if they were a single application with a double-click. You also might want to try **Alias Dæmon**, which allows to you to choose where on your system a newly created alias is to go. You just hold down the option key when making an alias to get a pretty dialogue box allowing you to select the target folder. Both of these can be found on `vader.eeng.brad.ac.uk` under `/pub/utils` as `qucklnch.hqx` and `alisasst.hqx` (dontcha just love those DOS 8.3 filenames). Both of these are shareware (but incredibly cheap, at only £3/US$5 each). Continuing in the alias vein, there's also **Alias Module**, which does much the same thing as Alias Dæmon but uses Ammon Skidmore's Extensions Strip to work its magic, and **Alias Exterminator**, which hunts down and ruthlessly slaughters dangling aliases. They're on my ftp site as `alismodl.hqx` and `alisxtrm.hqx` respectively.

All of the above should be available from the Mac archive site at sumex-aim.stanford.edu, and from any of the info-mac mirrors.

There’s also a small selection of joke programs to annoy your colleagues under `/pub/macshens`. These arose out of a discussion on the Usenet group alt.shenanigans about the dearth of good computer practical jokes. They’re harmless, but scary. Be prepared to be torn limb from limb if you use them on your friends/workmates. Share and enjoy!

Tiny Cipher Mailing List

If you would like to be informed of upgrades to Tiny Cipher, then send me an email and I will add your name to a mailing list. As soon as something new happens (like a maintenance upgrade is released), you’ll get an email from me telling you what’s going on. To subscribe to the mailing list, send an email to `david@vader.eeng.brad.ac.uk`, with the subject “`subscribe tiny cipher`”. The message body will be ignored.

D I S C L A I M E R

This software item (Tiny Cipher 1.5) is provided AS IS, without any warranty, even the implied warranty of MERCHANTABILITY or FITNESS FOR A PURPOSE. Any and all use is entirely at your own risk; the author accepts no responsibility with respect to this software item for any direct, indirect, consequential or incidental damage or loss of data (including damages for loss of business profits, business interruption, loss of business information, and the like) that may arise through the use or misuse of this software item. No representation, either express or implied, is made by the author as to the fitness of this item for a particular purpose, and no responsibility shall be accepted for any loss or damage arising from any defect in this product or its description, even if the author has been made aware of the possibility of such loss or damage.

The author (David A. G. Gillies) asserts the moral right to be identified as the author of this software item.

Version History

1.5 Introduced new password hashing routines. Added new version of TEA to combat minor weaknesses. Fixed a subtle bug that could (non-fatally) corrupt resource forks of files on decryption. Added Internet Config support for online manual and clickable URL in About… box. Several user interface inconsistencies fixed. Added progress box support for lengthy secure deletions.

1.4.2 Fixed a crashing bug introduced in v1.4 where occasionally a write would occur to invalid memory.

1.4.1 A bug fix to remove the requirement for the Navigation library. Sometime between the beta version and the final release this library became marked as a ‘strong import’, meaning that it had to be present for Tiny Cipher to run. It is no longer necessary (but it’s highly recommended).

1.4 Quite a big upgrade. Teacrypt compatibility added. Also added the ability to define the suffix appended to encrypted files. Full Balloon Help added for the first time. Added Appearance Manager support to give the MacOS 8 platinum look in dialogues and alerts, and support for the new Navigation Services (Apple's replacement for the rather elderly Standard File package).

1.3.2 Fixed a crashing bug on 32-bit dirty machines (e.g. Plus, SE/30) running pre 7.5 systems.

1.3.1 The first version accelerated for Power Macintosh. External differences there are none: I did some internal tweaking to handle the very different runtime architecture of the PowerMac (but very little - I'd designed the main code right from the start to port easily and in the end it took about an hour). Oh, and I recoded the encipher and decipher functions in PowerPC assembler. It gives about a 10% speedup over the C version.

1.3 A couple of bug fixes. Allowed secure folder deletion for the first time. Added the filename scrambling features and password re-entry. First version of the Apple Guide files.

1.2.1 Fixed a bug introduced by the folder processing additions of v1.2. If the auto-delete option was on, then only half the files in the folder would be processed, because the deletion was occurring immediately after the encryption or decryption and therefore the contents of the folder were changing, confusing Tiny Cipher. Deletion is now deferred until after all files have been processed, which cures the problem.

1.2 Fixed a few stupid and embarrassing bugs (whoops): fixed intermittent crash in or after password dialogue box, fixed folder processing to handle name collisions, removed lingering debugging code (double whoops), updated About… box. Added the facility to have passwords shorter than nine characters. Added the plain text password echo. Altered directory dialogue box to allow choice of a folder for encryption or decryption.

1.1 Fixed the folder encryption routines to allow drag and drop of an entire folder full of files onto Tiny Cipher's icon. Fixed the disk space checking routines to check that there is adequate free space on the target volume before starting.

1.0 Baseline version.

On The Horizon

Full AppleScript capability will be provided in version 2.0. If I can find time, there will be an archiving facility (somewhat like Stuffit) to group multiple files into a single, secure archive. This is likely to be a lot of work, so it might not make it to the first version 2 release. I write Tiny Cipher for fun (not profit) so the amount of time I can devote to it is limited by pressures of (paying) work.

Technical Stuff

This is some technical snippets I have assembled for your edification and enlightenment. Skip this section if you are easily bored.

Here’s ANSI C source for the (old-style) Tiny Encryption Algorithm, in case you were wondering:

`delta` is a magic number chosen to be the integral part of the Golden ratio $\sqrt{5/4} - 1/2$ multiplied by 2^{32}. `n` is the number of rounds.

```
void Encipher(const unsigned long *const v,unsigned long *const w,
   const unsigned long *const k)
{
   register unsigned long a=k[0],b=k[1],c=k[2],d=k[3],y=v[0],z=v[1],
      sum=0UL,delta=0x9E3779B9UL,n=16UL;

   while(n--)
      {
      sum+=delta;
      y+=((z<<4)+a)^(z+sum)^((z>>5)+b);
      z+=((y<<4)+c)^(y+sum)^((y>>5)+d);
      }

   w[0]=y;
   w[1]=z;
}

Note that for a given number of rounds, n, sum = delta*n.

void Decipher(const unsigned long * const v,unsigned long * const w,
   const unsigned long *const k)
{
   register unsigned long a=k[0],b=k[1],c=k[2],d=k[3],y=v[0],z=v[1],
      sum=0xE3779B90UL,delta=0x9E3779B9UL,n=16UL;

   while(n--)
      {
      z-=((y<<4)+c)^(y+sum)^((y>>5)+d);
      y-=((z<<4)+a)^(z+sum)^((z>>5)+b);
      sum-=delta;
      }

   w[0]=y;
   w[1]=z;
}
```

This is where the algorithm gets its name. By comparison, DES is several hundred lines of C. TEA is beautifully suited to software implementation on a 32 bit microprocessor with a fair number of general purpose registers (like a 68xxx or PowerPC). It’s not quite as efficient as SEAL, but it’s still very compact. It wouldn’t be too difficult to shoehorn into an ASIC or FPGA, either.

The newer version of TEA fixes a couple of very minor weaknesses discovered by David Wagner of Berkeley in California. These weaknesses are unlikely to be exploitable in real life, but for peace of mind I implemented the updated algorithm in version 1.5. There are two fixes to the algorithm. One is to adjust the key schedule so that all 128 bits of the key are fully involved in the encryption process, and the second is to introduce the key material more slowly. This does not hurt the speed of the algorithm — in fact on a per-round basis the new variant is if anything slightly faster. In the new algorithm I use 24 rounds for added security, so the raw processing speed is slightly less (by about 10%) but I/O bandwidth issues still dwarf raw encryption throughput. The new algorithm is, like the older version, simplicity itself:

```
void Encipher(const unsigned long *const v,unsigned long *const w,
   const unsigned long *const k)
{
   register unsigned long y=v[0],z=v[1],
      sum=0UL,delta=0x9E3779B9UL,n=24UL;

   while(n--)
      {
      y+=(z<<4 ^ z>>5) + z ^ sum + k[sum & 3];
      sum+=delta;
      z+=(y<<4 ^ y>>5) + y ^ sum + k[sum>>11 & 3];
      }

   w[0]=y;
   w[1]=z;
}


void Decipher(const unsigned long * const v,unsigned long * const w,
   const unsigned long *const k)
{
   register unsigned long y=v[0],z=v[1],
      sum=0xD5336958UL,delta=0x9E3779B9UL,n=24UL;

   while(n--)
      {
      z-=(y<<4 ^ y>>5) + y ^ sum + k[sum>>11 & 3];
      sum-=delta;
      y-=(z<<4 ^ z>>5) + z ^ sum + k[sum & 3];
      }

   w[0]=y;
   w[1]=z;
}
```

Implementation Details

The actual code used in Tiny Cipher is written in 68000 and PowerPC assembler for optimum speed, but the gain over the C implementation is actually quite marginal (in fact, a good optimising compiler will probably give you code that is just about as fast as you can get by hand). I profiled TEA on a Hewlett Packard 9000/730 workstation (a Unix box with a fairly fast 32 bit PA-RISC 7100 processor) and found that unrolling the while loop to a depth of four gave about a 17% speedup. Any more than that and the code filled up an entire cache line, so that the cache was getting blown every time round the loop. The fact that TEA is a Feistel cipher (where the output of the previous round is split in two and swapped round before re-entry to the cipher function) means that the possibilities for instruction re-ordering are very limited. The instructions really do have to be carried out in exactly the given order. The biggest gain comes from giving all the variables register storage class and putting v and k into local variables. On the Hewlett Packard this speeded things up by about three times! This is similar on a PowerMac where the most expensive operations are memory accesses. The Metrowerks CodeWarrior compiler I used to build the latest version of Tiny Cipher has a very clever optimiser, and it took some doing to get my hand-rolled assembler to beat the compiler by more than 10%.

There is a trivial speedup that I did not bother to implement: the first addition to sum in encipher, and the last subtraction in decipher are redundant. They can be taken out of the loop thus;

```
sum=0x9E3779B9UL; /* instead of 0 */
n=15UL; /* instead of 16 */

y+=((z<<4)+a)^(z+sum)^((z>>5)+b);
z+=((y<<4)+c)^(y+sum)^((y>>5)+d);

while(n--)
   {
   sum+=delta;
   y+=((z<<4)+a)^(z+sum)^((z>>5)+b);
   z+=((y<<4)+c)^(y+sum)^((y>>5)+d);
   }
```

and similarly for decipher. This may even be slower on a processor with a large cache and dedicated looping hardware.

If you absolutely have to get the last ounce of speed out of your code, then I guess an assembler implementation is the only way to go. On the PowerPC, it's the only way you'll be able to exploit hacker tricks like branch prediction hinting and re-ordering in the absence of a really good optimising compiler.

For the new variant of TEA, there is a noticeable (i.e. measurable) speedup if the four portions of `key` that are used in each round are hardwired. For example, in the first four rounds, `k[0]`, `k[1]`, `k[2]` and `k[3]` are added to `y`, whereas `k[3]`, `k[2]`, `k[1]` and `k[0]` are added to `z`, in that order. It makes for larger code, but still small enough to fit in the L1 cache on a PowerPC. Even if it gets blown out of the L1 cache by something else, it'll be in L2 cache. New variant TEA with 24 rounds benchmarks at 64 Mbits s^{-1} on my 8600/250 in ECB mode. The speedup comes from the fact that memory access is slow. Even accessing the L1 cache is not a zero wait-state operation. If data has to be fetched from main memory, then many cycles can be wasted. And if virtual memory is enabled, and a page fault occurs, the penalty is measured in millions of lost cycles.

TEA can be operated in any of the four standard modes. Tiny Cipher uses **Cipher Block Chaining** (CBC), which is why encrypting the same file twice in a row will give a very different ciphertext. CBC circumvents the phenomenon known as **stereotyped beginnings and endings**, where two files with identical plaintext blocks will encrypt to the same ciphertext under identical keys. If you know that a certain plaintext block encrypts to a given ciphertext block, then every time you see that block in the ciphertext you can decrypt it, even if you don't know the key. Files will often have very similar data at the start and end of files (file headers/footers, for example) and simply using a block cipher in Electronic Code Book mode helps an attacker to gather statistics to start a cryptanalysis. CBC also thwarts a nasty technique known as **block replay,** where an attacker can substitute blocks of his choosing in a ciphertext to alter the underlying plaintext when the receiver decrypts the message. By chaining (i.e. feeding back) the output of one encryption into the next, every block in the ciphertext depends on every previous one. If the first block of the plaintext is random data, then two otherwise identical messages will give different ciphertexts. This random block is called the Initialisation Vector, and it doesn't even need to be kept secret. As long as it changes every time a file is encrypted, all is well. Tiny Cipher stores the initialisation vector in its resource fork as the resource of type ‘TeaV’, ID 128. This is why Tiny Cipher must be run from an unlocked volume. Don’t alter this resource!

Key generation in v1.4 and earlier used my own fairly ordinary bit-expansion routines followed by a tandem Davies-Meyer hashing (using TEA as the block cipher!) to hash this to the 128 bit key. I’ve been careful to lock all the memory associated with the key into physical RAM to stop virtual memory paging it out to disk. I always overwrite the memory in question before unlocking it, as well. The new password-to-key generation scheme uses the Tiger secure hash algorithm invented by Ross Anderson and Eli Biham, which completely fixes any problem of password collision.

Cryptanalysis

Here’s a little bit of hand-waving, non-rigorous analysis of old-style TEA:

First let’s look at the key-schedule of TEA. The choice of delta, while fairly arbitrary, turns out to be very good. It is a 32 bit number, so the values that `sum = k*delta, k=1,2,3,`… takes on as the algorithm goes on must be reduced modulo 2^{32}. It’s obvious that `delta` is coprime to 2^{32}, so as `delta` is successively added to sum it generates all the numbers between 0 and 2^{32} - 1 (of course not in any apparent order). Since the ratio of `delta` to 2^{32} is the golden ratio, the values of `k*delta` ‘fill in’ the numbers between 0 and 2^{32} - 1 in an optimally spread out way (this is analogous to the golden section root finding search). The main thing is that for any sensible number of rounds (<several thousand, say), `sum` takes on a very different value (in other words, its bit pattern is very different) in each round.

TEA uses two operations taken from orthogonal arithmetic groups (XOR and addition) to form a weakly non-linear function. This is iterated a number of times to ensure adequate diffusion. The top four and bottom five bits of a 64 bit ciphertext block that has been through one round are the weakest, which is why at least 6 rounds are required to diffuse the plaintext adequately. But how many rounds are enough? We know that any product cipher that is iterated long enough is going to give us the alternating group, but we would like to have the most efficient algorithm possible (i.e. keep the number of rounds to the minimum consistent with security). My feeling is that 16/24 rounds, as used in Tiny Cipher, is more than adequate, and it’s still pretty fast. For applications where the data age very fast (like real-time video) then eight rounds is probably sufficient, and that can give you gigabit-per-second throughput with modest hardware requirements (like XiLINX or Altera chips). Teacrypt uses 32 rounds.

Note that TEA is a new cipher. It doesn't have the track record of the more well-known block ciphers like DES or IDEA. No-one has demonstrated, or even tackled as far as I know, the question of whether TEA is a group (this is a very technical question to do with the utility of multiple encryption). I think it's pretty unlikely that TEA is a group; if not then triple encryption is useful since the effective 256-bit key makes a meet-in-the-middle attack computationally infeasible for effectively all time.

Technical References

Probably the best source of information on cryptography for the general user is Bruce Schneier’s superb book, **Applied Cryptography**, 3rd Ed., published by Wiley. Another excellent book is Dorothy Denning’s magisterial work, **Cryptography and Data Security,** published by Addison-Wesley. For the layman or semitechnical user, these two books should give you all the information you need. Of course, they contain sufficient references for a lifetime of study. Lurking around the `sci.crypto` and `sci.crypto.research` Usenet groups is a good way to learn, too. There's lots of information about cryptography out there on the Net. Of course you have the usual problem with net-borne data of deciding whether it's utter rubbish or not. But the University sites devoted to cryptography are usually a mine of useful information. The Cambridge and Oxford sites are both very good.

A Frequently Asked Question

Is there a version of Tiny Cipher for the IBM PC?

The short answer is no. The long answer is also no. I find the whole subject of Microsoft's pathetic attempt at producing an operating system a nightmarish, hive-inducing, productivity-sapping no-go area. The Windows NT server box that I am being forced to develop software on at the moment (for my real job) is one of the most useless hunks of junk I have ever encountered. The basic hardware is fine (266 MHz PII), but the OS looks like beta version software. This protected-memory, pre-emptively multitasking operating system is less stable by about two orders of magnitude than my Mac. It can't redraw the screen properly, the screen saver causes a crash bad enough to need a hard boot, and Microsoft's Internet Information Server has half the functionality of the Apache web server. So the likelihood of a PC version (let alone a Windows [sic] version) appearing is roughly akin to the likelihood of my undertaking "gender reassignment surgery" and then winning the Miss World competition. And if you saw what I look like in heels then you would realise that that is an extremely slim chance indeed…

There **is** an implementation of TEA for Unix machines (i.e. the other variety of proper computers). It's called teacrypt(1) and it's available in binary form for HP-UX version 9.05 and later, and Solaris 2 or later from my ftp site at `ftp://vader.eeng.brad.ac.uk/pub/crypto/`

Note it uses the new-style opendir()/readdir() functionality in favour of the old BSD scandir() routines so it should be fairly portable.

If I can get permission from the author of the random number generator I use to do the key generation then I will publish the source code (or I might switch to using Tiger/128, although this would mean backward compatibility would be lost). Teacrypt is a much simpler piece of software than Tiny Cipher — it's only a few hundred lines of C.

Note that versions of Tiny Cipher earlier than 1.4 cannot read or write teacrypt files. I might port teacrypt to the PC as a DOS application, but I am not going to spend the time required to learn Windows in all its various Satanic incarnations and then port a 35 000 line program unless I get paid the going rate (£30/hour). If someone wants to license the technology for the PC then that's fine by me too. You might be able to cook up a front end with the same look and feel as the Mac version in Visual C++ or even MFC, and then plug the encryption routines in, but if this happens it won't be me doing it.

The Without Whom Department…

Sir Roger Needham FRS
Dr. David Wheeler — the guys who invented TEA. Heroes!

Dr. Simon J. Shepherd — Mr. Cryptography at Bradford Uni., all round sound bloke and beer monster. One of the top cryptographers in the U.K., he was the one who got me started in the whole field.

Dr. Bo Lin — Now at Motorola Smart Card Applications. Very, very smart. He was the one who pointed out the strength of TEA's key schedule.

Dr. Matthew G. Parker — this guy is so clever it frightens me. Bradford Uni's best number theorist (now at the University of Bergen), he helped firm up a lot of the ideas I had about the security of Tiny Cipher.

Many thanks also to David C. Oshel, beta tester extraordinaire, who tirelessly tracked down numerous bugs and inconsistencies, and then helped me mercilessly squish them. Tiny Cipher is a better product for it.