TurboCrypt

7.8

© 1999-2005 PMC-Ciphers, Inc, All Rights Reserved

Inhaltsverzeichnis

TurboCrypt V7.8

PMC-Ciphers, Inc. 2005

Welcome to TurboCrypt - Ultra-secure Encryption Suite.

This document provides details about using TurboCrypt to create, manage and use file hosted volumes, sending e-mails with encrypted attachments, encrypting files and folders, securely wiping files, folders, as well as unused disk space and removing traces from your PC.

Content

-o-

Backround Information and White Papers

TurboCrypt

TurboCrypt Security Suite: White Paper
Short White Paper about TurboCrypt.

Disk Encryption: Mode of Operation
... how TurboCrypt works ...

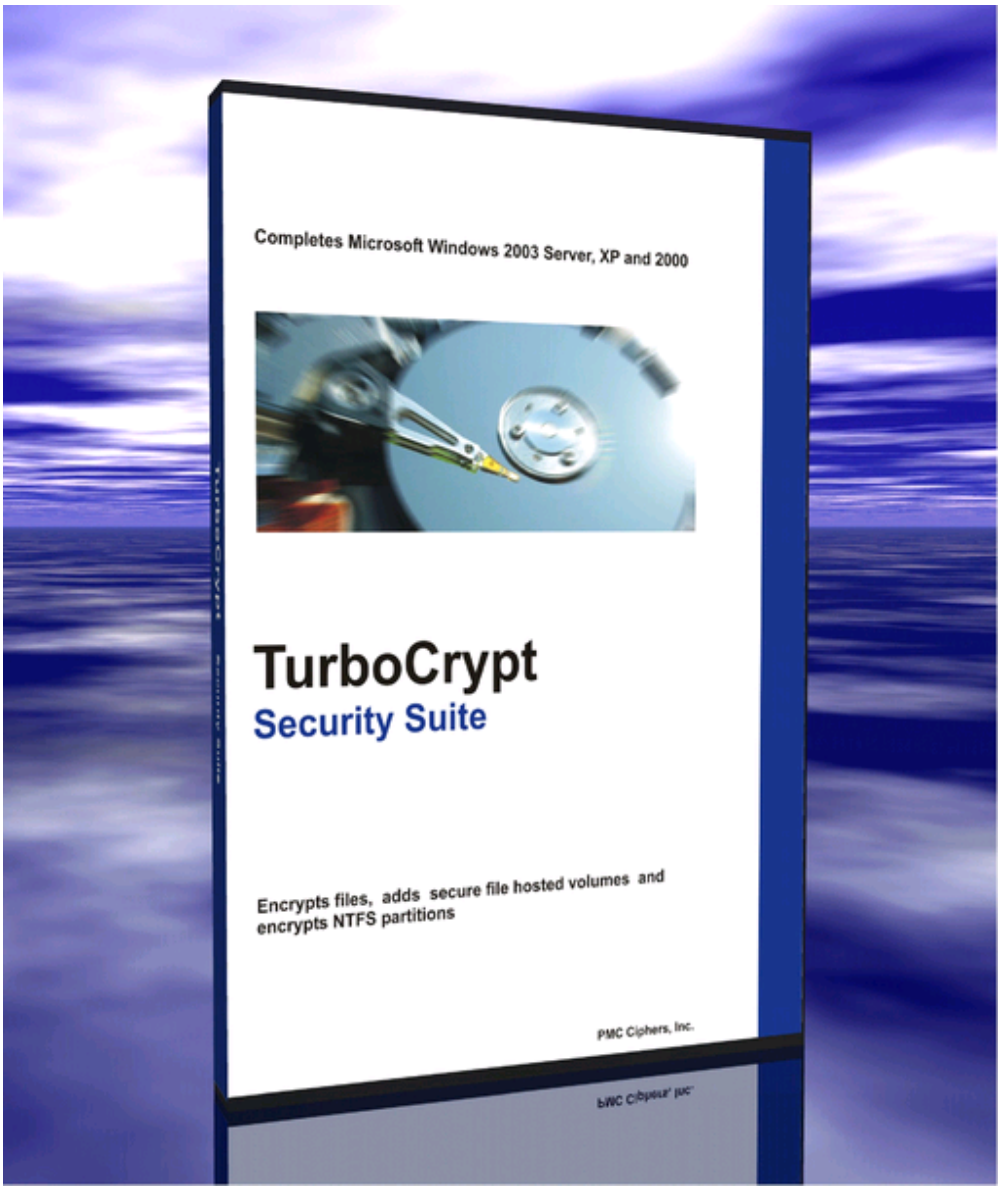
Polymorphic Cipher

The Polymorphic Cipher
Description and history of the polymorphic cipher (not "too" technical).

Fact Sheet 256 and 512 bit Encryption
TP2 (TurboPMC) data sheet.

Diehard Randomness Test Suite - Test Results
Diehard Randommness Test on the 512 bit Polymorphic Block Cipher imlemented in the Polymorphic Cipher version of TurboCrypt.

-o-



PMC Ciphers, Inc. TurboCrypt Security Suite: Ultra-secure Data Encryption and Privacy Protection

White Paper

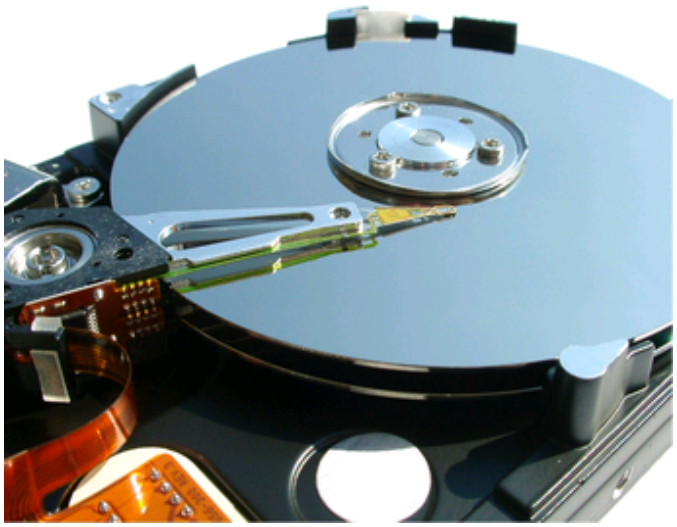
First published: December 2002, latest revision: June 2005

For the latest information, please see http://www.pmc-ciphers.com

Introduction

The simple password protection mechanisms of popular Office packages or compression utilities can generally be broken or bypassed easily. Additionally, a multitude of files and links that are accessible to every user reveal often sufficient information about a user's habits and secrets.

In order to counteract, TurboCrypt provides file encryption, secure wipe and trace deletion functionality through the context menu on right mouse click in Windows Explorer.
TurboCrypt further supplies encrypted disk drives (file hosted volumes) which can be mounted at system start or any time later.
These encrypted logical volumes are fast, seamless, integrated, and come with ultra-high security 512 bit PMC encryption or alternatively FIPS-197 compliant AES encryption using two separate 256 bit AES engines.



All the structures needed by your operating system to recognise a file system of a particular type, such as FAT or NTFS, are made available through the TurboCrypt encryption driver. The plug-and-play TurboCrypt encryption driver has been programmed especially for Windows NT5.X (Windows 2000 and XP and later).

TurboCrypt volumes can be as big as 2TB!!! (1TB = 1000GB). The Enterprise Edition of TurboCrypt is even capable of encrypting and controlling physical NTFS partitions!
To your operating system, these new volumes look exactly like your A: or C: drive, or any other volume on your computer.

The implemented realtime crypto engine is based on the Polymorphic Encryption Algorithm (PMC) invented by PMC Ciphers. Its encryption speed outperforms AES (Rijndael algorithm) by factor 10 in terms of speed and by factor two in key length! TurboCrypt takes advantage of the probably fastest crypto engine in the world and generates ciphertext with perfect randomness.
The AES version of TurboCrypt is fully FIPS-197 compliant. AES test vectors are automatically checked by the control panel. The AES crypto engine simulates 512 bit operation by using two separate 256 bit AES algorithms. Passwords are generated

using SHA-1 and MD5.

The Control Panel software of TurboCrypt manages encrypted volumes and partitions. New encrypted drives can be formatted, mounted, unmounted and removed to/from the file system at any time.

Significantly Enhance Privacy

Privacy of existing software installations in offices is generally poor. Notebook computers can get lost. Vital financial data or technical details of new developments can thus be stolen. Data is generally transported as plaintext in e-mail attachments.
By storing important data of your company on an encrypted TurboCrypt volume, the immanent risk potential is minimized.
File Encryption further provides security for files that are sent via e-mail. Files and folders are encrypted at a touch of a button through the OS shell (Windows Explorer). It is even possible to create self-extracting encrypted archives and to send them via e-mail to users who haven't installed TurboCrypt. All required functionality to extract and decrypt these archives is provided with the self extractor. The TurboCrypt selfextractor also runs on Win98 or Win NT machines.

Additional trace deletion and secure wipe functionality increase privacy:

- Secure wipe of files, folders and unused disk space render forensic analysis of a hard disk an impossible task.
-

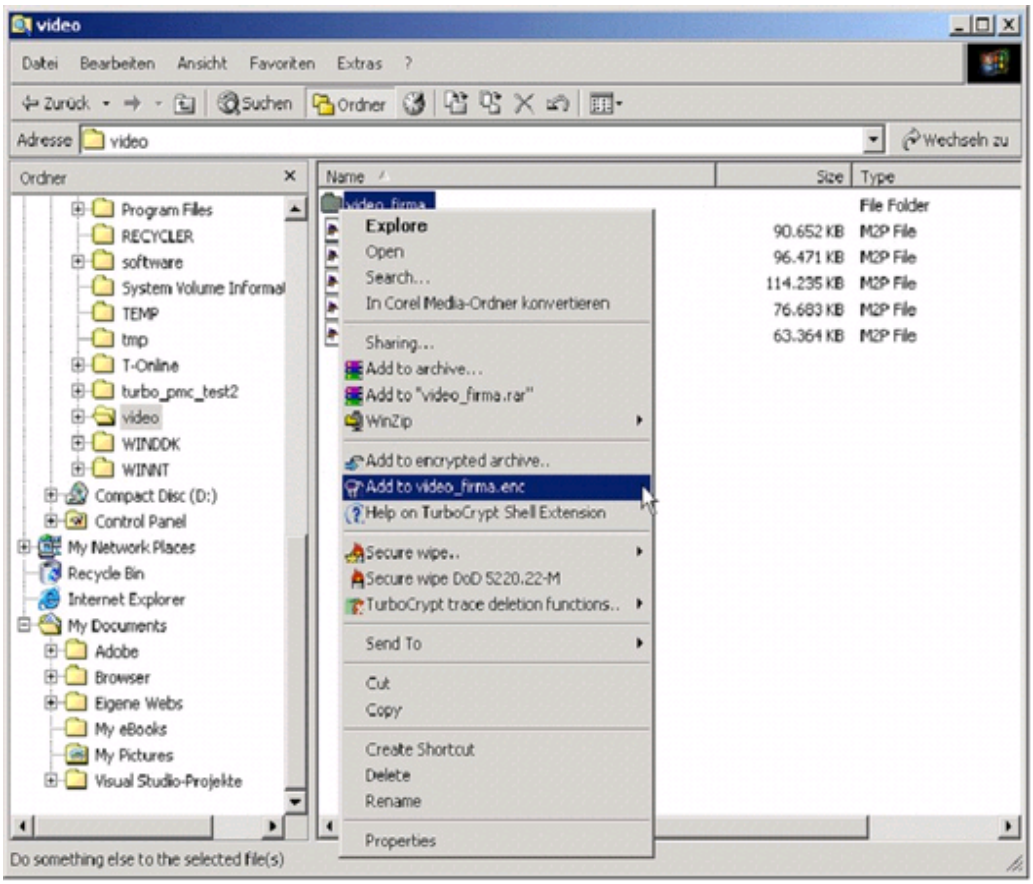
- Trace removal features the following functions: Deletion of cookies, internet history list, Internet Explorer cache, recently used document list, temporary files, Media Player recently used files list, Office MRU lists, Office recent files in personal folder, FrontPage registry traces, CDBURN personal folder, etc.
  Automatic download and launch of the most up-to-date trace deletion extension utility is provided to complete trace deletion functionality. Currently this tool cleans traces left by use of the RealPlayer.

Ease of Use

Usability studies indicate that users are easily overwhelmed by complex interfaces. This conclusion is supported by a large amount of interviews. A simplified user experience aids in decreasing support costs, while making users of all levels more efficient and productive.

TurboCrypt is absolutely easy to use.

As an example, the encryption driver is automatically installed when the software is used for the first time. New versions automatically update the environment.



TurboCrypt provides file encryption through the context menu on right mouse click in Windows Explorer.
The file encryption integrates like other popular tools like WinZip or WinRar in a very ergonomical way.
Additionally, files and folders can be securely wiped using three different methods: Fast wipe, DoD 5220.22-M and Gutmann.
Secure wipe of unused disk space is provided as well.

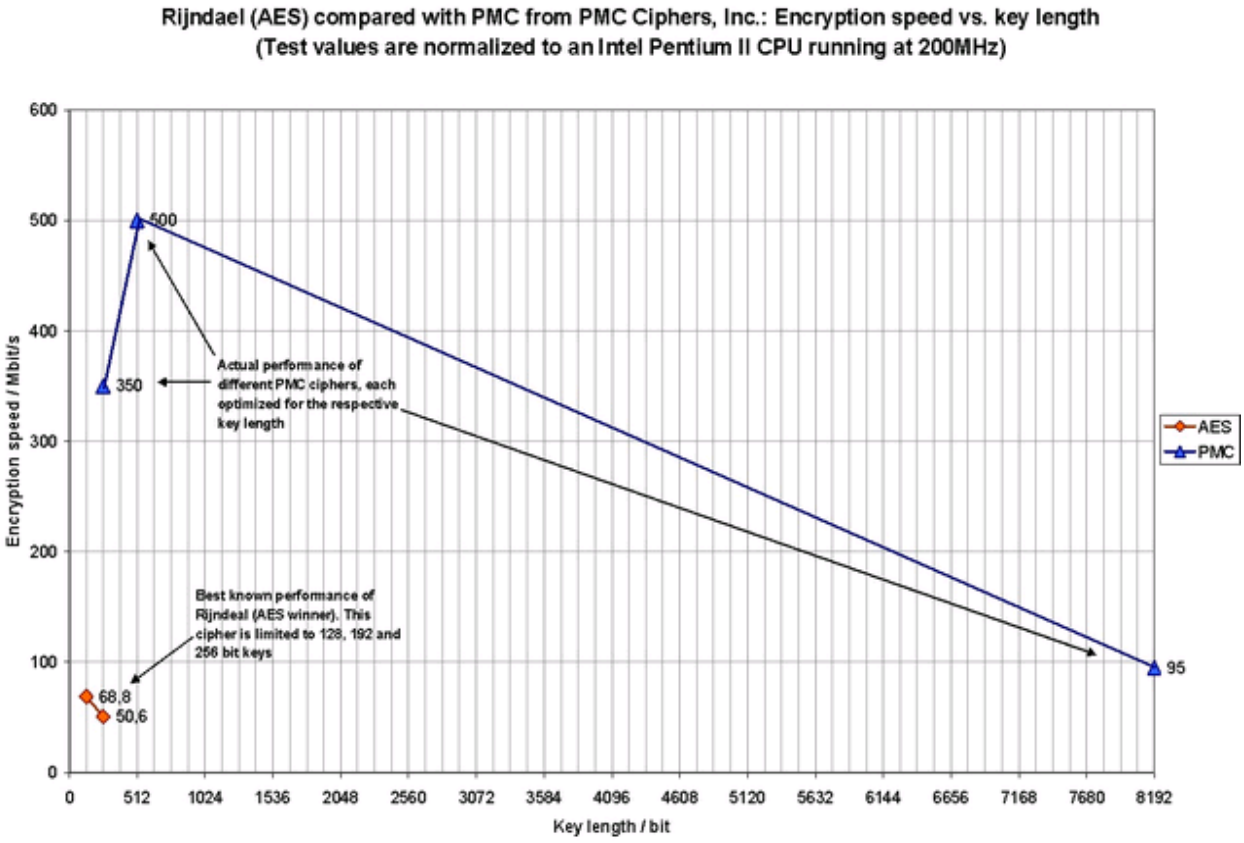
Latest Technology

Latest Encryption Technology for unprecedented security. TurboCrypt takes advantage of self-compiling crypto code, one of the latest achievements in the science of cryptography. Polymorphic Cryptography has been a state secret in Germany until 1999.
Security and processing speed are no more contradictory. There exists no attack other than trying every possible key combination. There are exactly $1.34078079299 * 10^{154}$ such key combinations ($1.34078079299 * 10^{154}$ = 13407 8079299000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000). The implemented crypto engine is fast enough for real-time video playback. It encrypts/decrypts data at approx. 5GBit/s on an AMD Athlon XP1800+ microprocessor.

The AES version of TurboCrypt implements two 256 bit Rijndael algorithms that encrypt 2x256 bit in order to be compatible to the 512 bit environment that stems from the polymorphic version of TurboCrypt. A fast table-based AES implementation was chosen in order to make the algorithm as fast as possible. 256 bit AES test vectors are checked frequently in order to make sure that the algorithm has not been corrupted. Password hashing is performed using SHA-1 and MD5.

The AES standard (FIPS-197) can be found here:
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

PMC Ciphers, Inc. does not guarantee that the AES standard is secure as the paper above which has been approved by the U.S. Secretary of Commerce states: "6. Applicability. This standard may be used by Federal departments and agencies when an agency determines that sensitive (unclassified) information (as defined in P. L. 100-235) requires cryptographic protection."
The following question remains: How to encrypt classified information?



Conclusion

- TurboCrypt introduces key functionality which greatly reduces the risks that unauthorized personnel and organizations gain access to your company's data.
- Vital company data remains secret in case a notebook computer is lost or stolen.
- Existing NTFS disk partitions and external hard drives can be fully encrypted.
- Virtual drives and big hard disks up to 2000 GB (!) are easily encrypted with TurboCrypt Encryption.
- Ultra-secure and ultra-fast 512 bit PMC encryption technology is so fast that users cannot feel any speed difference.
- The AES version of TurboCrypt uses FIPS-197 compliant 256 bit AES (Rijndael) encryption.
- Secure wipe of files, folders and unused disk space render forensic analysis of a hard disk an impossible task.
- Deletion of cookies, internet history list, Internet Explorer cache, recently used document list, temporary files, Media Player recently used files list, Office MRU lists, Office recent files in personal folder, FrontPage registry traces, CDBURN personal folder, etc.

For more information: http://www.pmc-ciphers.com

-o-

PMC Ciphers, Inc. TurboCrypt
Security Suite

Disk Encryption: Mode of Operation

Technical Paper

First published: August 2003, revised in June 2005

For the latest information, please see http://www.pmc-ciphers.com

Introduction

TurboCrypt provides encrypted disk drives (NTFS partitions) and encrypted file-hosted volumes to Microsoft Windows NT5.X file systems. The software relies on a kernel mode encryption driver which is added to the Windows disk driver stack at system boot.

To your operating system, encrypted TurboCrypt volumes look exactly like your A: or C: drive, or any other drive in your system.

The encryption system at boot time of the operating system

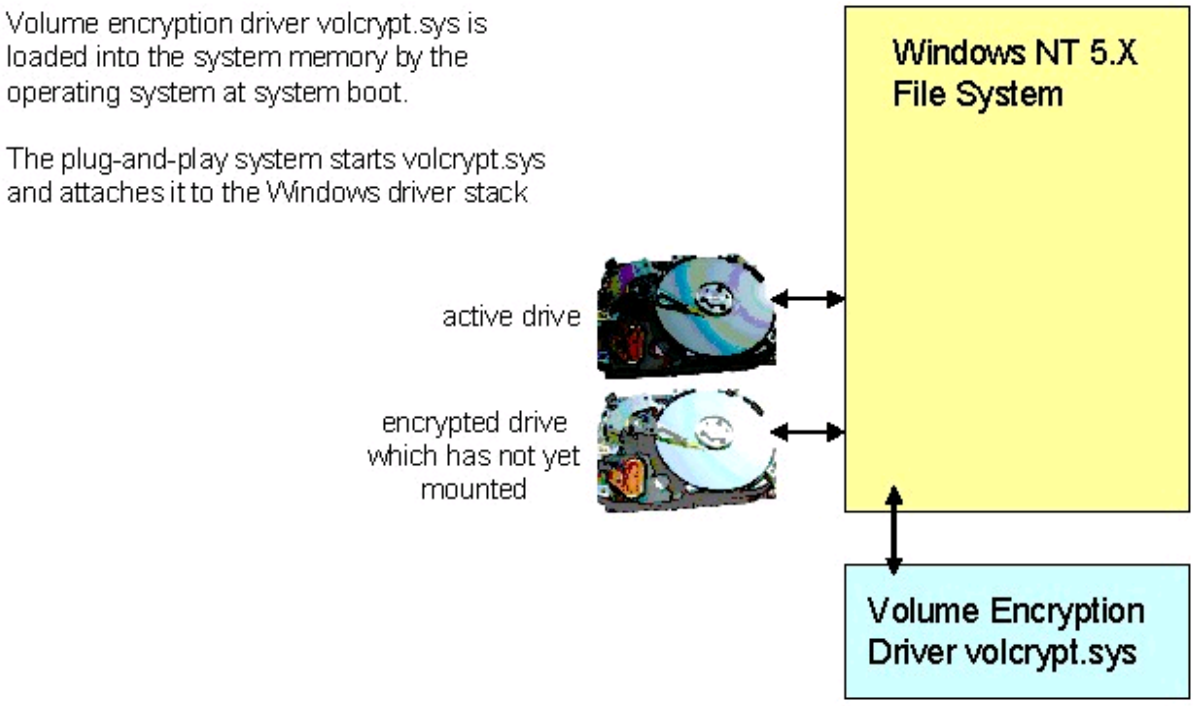
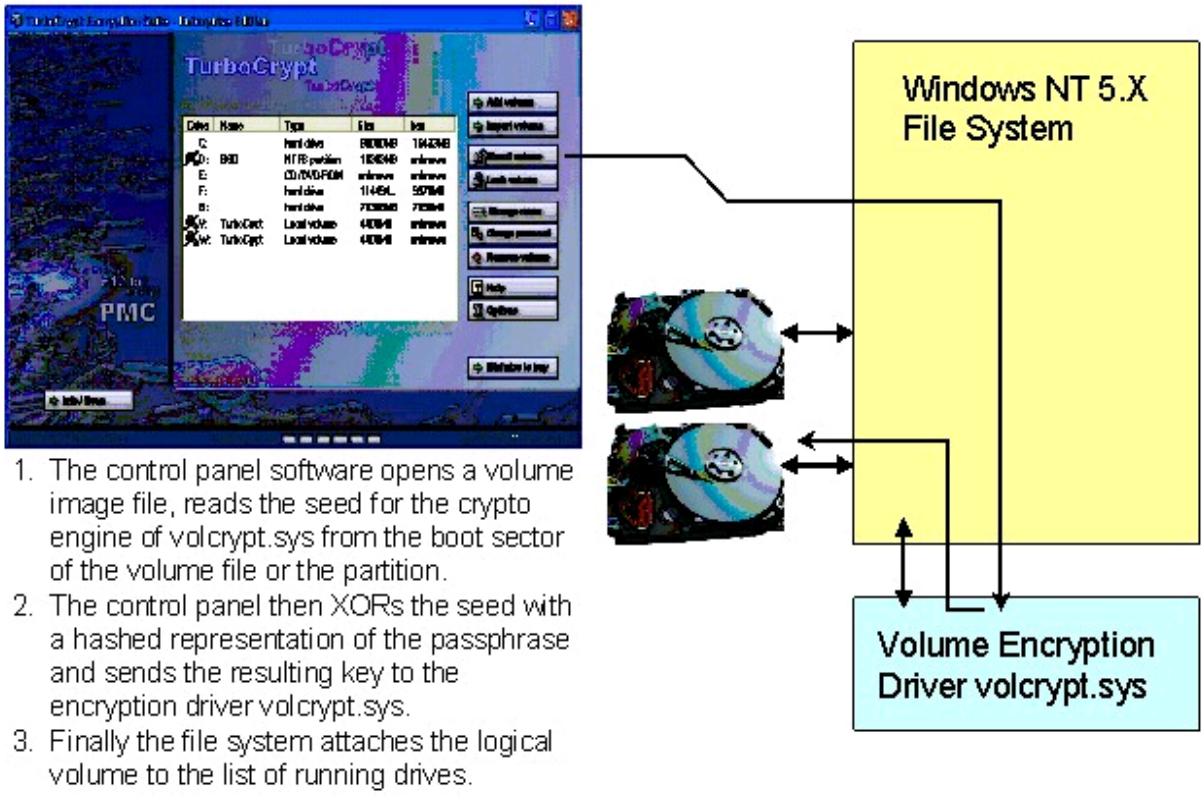


Fig. 1 System boot with the encryption driver

Runtime operations



1. The control panel software opens a volume image file, reads the seed for the crypto engine of volcrypt.sys from the boot sector of the volume file or the partition.
2. The control panel then XORs the seed with a hashed representation of the passphrase and sends the resulting key to the encryption driver volcrypt.sys.
3. Finally the file system attaches the logical volume to the list of running drives.

Fig. 2 Attaching a TurboCrypt volume to the NT 5.x file system

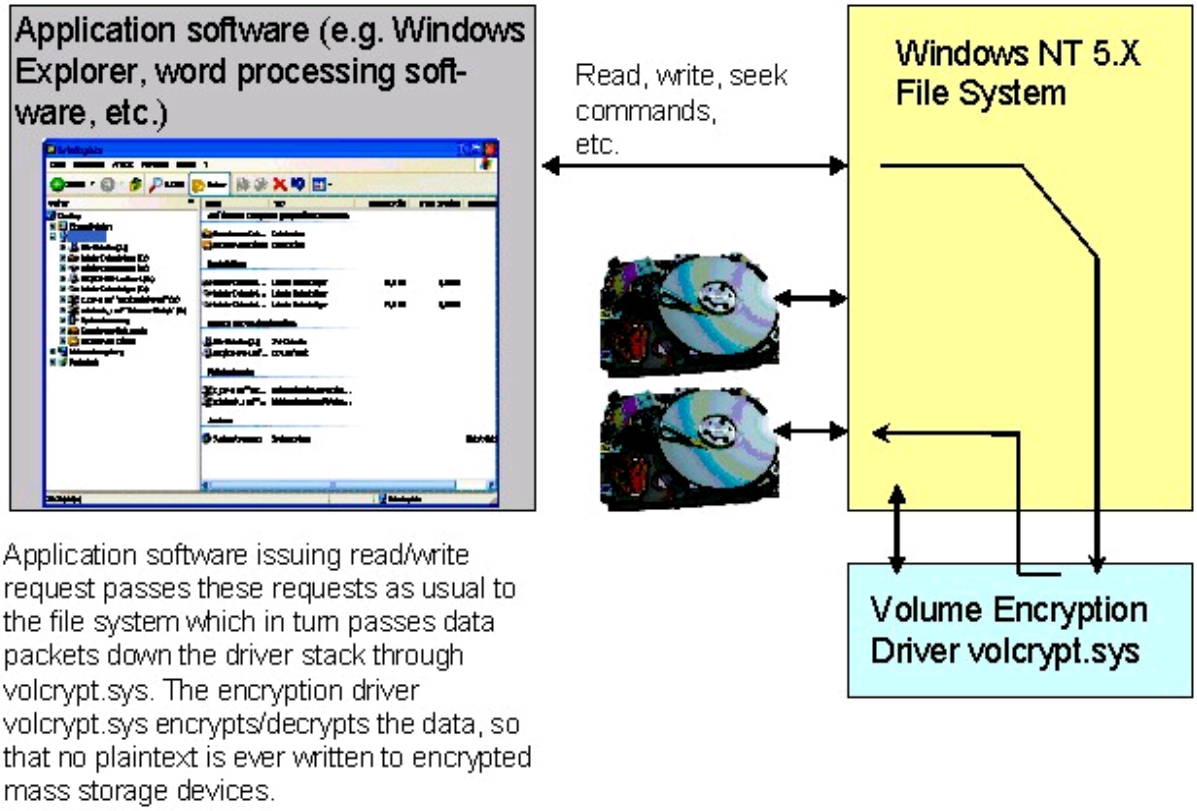
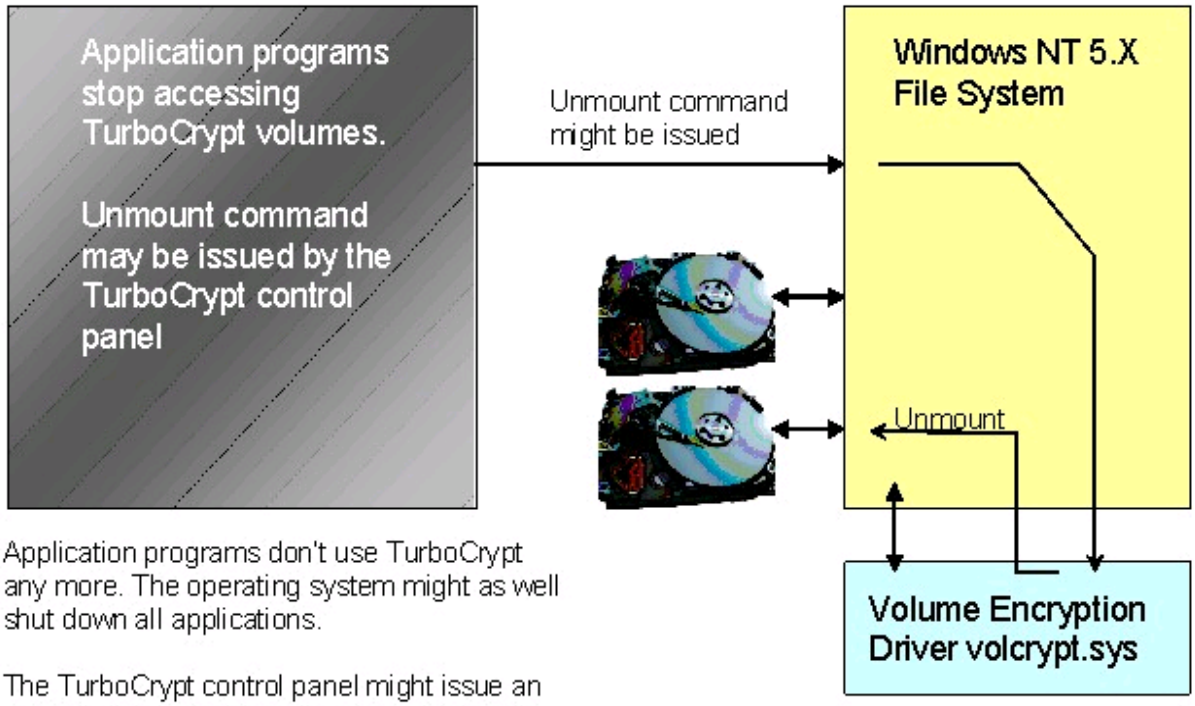


Fig. 3 Normal Read/write operation to a TurboCrypt volume/drive

Service termination and shutdown



Application programs don't use TurboCrypt any more. The operating system might as well shut down all applications.

The TurboCrypt control panel might issue an unmount command to specific TurboCrypt volumes or all TurboCrypt volumes.

No file handles to TurboCrypt volumes are open any more.

The plug-and-play system may force all drivers to stop immediately. The file system flushes all buffers.

volcrypt.sys checks for open files handles. If there are no open handles, the encrypted TurboCrypt drives are unmounted. Plug-and-play IRPs might force immediate and unconditional shutdown.

Fig. 4 Shutdown of TurboCrypt volumes/drives

For more information: http://www.pmc-ciphers.com

Microsoft, the Office logo, Outlook, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.
Company and product names mentioned herein may be the trademarks of their respective owners.

-o-

The Polymorphic Cipher

Author: C. B. Roellgen
First release: 1999, revised 2000 - 2005

The history of cryptography has shown that unbreakable ciphers had in many cases been cracked shortly after their widespread use had begun. One famous example is the "Enigma" encryption machine used by the Nazis in world war two: British specialists at Bletchley Park had been able to crack the clever permutation code. This resulted in detailed knowledge about most German military operations and in the total loss of the German submarines.

The widespread DES algorithm has long been supposed to be unbreakable. In January 1999 a test performed by RSA Data Security, Inc. (San Mateo, Calif., USA) proved that it takes less than 22.25 hours to crack the 56 bit algorithm by brute-force (by trying all 2^{56} possibilities). That was 365 days after the same company needed 41 days for that task! RSA claim to have a much better cipher, which is obviously true.
Today a code length of 128 bit is regarded as safe by experts (1000 bit for RSA). Thus, it took just 25 years for the experts to actually double their requirements in key size (which is effectively 10000000000000000000 times more than what was initially regarded "safe"!!!).

It is indeed true that a key size of 256 bit are absolutely secure. There exists (even theoretically) no machine which can break such a code within an acceptable period of time. But one can never be sure that there exists absolutely no shortcut.

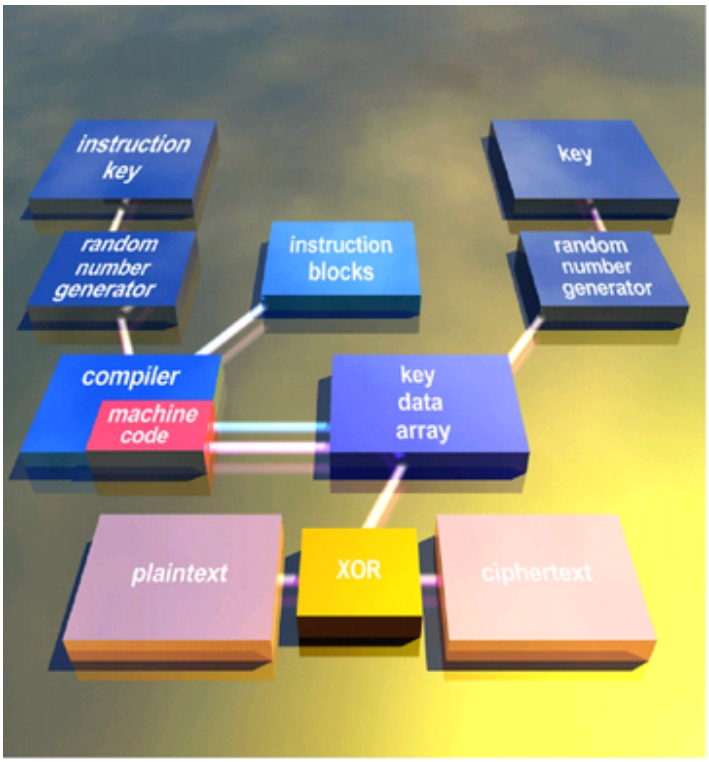
Technology advances quickly. Day by day it becomes more likely that somebody is able to decrypt files containing confidential data. Why not simply oversize the encryption algorithm to solve this nasty but inevitable security problem?
The main reason is speed! It is usually claimed that long keys slow down the algorithm too much. That's true because execution time increases at least by the key size at the power of two.

A new approach implying self-compiling machine code solves that problem. Execution time increases only linearly with key size. The idea behind it is to randomize the algorithm itself. That’s why I have named it „Polymorphic Method“. What if both data and the actual encryption algorithm are undefined in the beginning. An Opponent who wants to break your key feels deprived of any constant. Working with variables only quickly becomes pretty complex. Commonly known ciphers use one key - say one variable.

A mathematic equation comprising two variables cannot be solved! For cryptography, there is of course a solution - but the only way to find it is to search exhaustively the whole keyspace. This problem is one-dimensional for common ciphers and two-dimensional for the Polymorphic Cipher.

The Polymorphic Method is among the strongest ciphers available today and it's probably the strongest. The method simply takes advantage of machine code assembled at random to yield extraordinary security against all kinds of attacks. It is even intrinsically safe against the analysis of the program's instruction sequence because the instruction sequence itself is a variable! It is important to know that the key assumption for successful cryptoanalysis is detailed knowledge of the encryption algorithm - but the actual Polymorphic Method’s algorithm is inherently UNKNOWN.

Basic principle of the Polymorphic Method



Two different passwords (or two parts of one password) are fed into random number generators. The one RNG on the left produces a byte stream which is compiled into machine code. The compiler simply assembles standardized building blocks, adjusts addresses as well as entry- and exit points to generate a piece of machine code which affects the key data array during execution of the machine code. The key data array is initialized by the right RNG which is biased by the right password.
After the machine code has been executed, the content of the key data array can be used to encrypt plaintext through the application of the xor function. The content of the key data array can and should alternatively be used for biasing an underlying cryptographic algorithm which is simple and fast. By doing this, the complexity of the total crypto system increases and it becomes much more difficult to analyze the internal state of the key data array, although the information it contains gives no clue about the keys.

It is even more confusing to sometimes recompile the instruction sequence. This makes the method dynamically polymorphic.

The compiler internal to the Polymorphic Encryption Method compiles replaceable code fragments which use the processor's registers in an identical way. Each building block can be exchanged by any other. The actual code length can vary due to differences in complexity but not the way data is passed from one building block to the other. A data array is used as a long variable which is initialized by a password. It takes the place of the key as known from conventional crypto algorithms. The CPU works on this key data array and performs permutations, modulo-divisions, shifts and other nonlinear operations.

An implementation of a Polymorphic Method is publically available as a Windows program called „Best Possible Privacy". It's crypto engine uses the CPU register ebx as input and output register, eax as general purpose buffer and ebp as base pointer to the key data array. The key data array that ebp points at is 256 bytes long.

Example of a simple building block

The xor operation alters ebx and four bytes of the key data array:

```
push ebp;                  // save the start address of the key data array for later
mov eax,123;                  // load offset: constant data which was calculated by the compiler
add ebp,eax;
```

```
  mov eax,[ebp+0];         // load key[ebp+0] in AL and key[ebp+1] in the next upper byte of eax
and
                        // so on up to key[ebp+3]
   xor ebx,eax;                    // this instruction can be replaced by another or a set of
instructions
  xor [ebp+x],ebx;         // change the key data array frequently; x is defined by the compiler
and
                      // chooses one element of the key
  pop ebp;               // restore start address of the key data array
```

Instead of xor it is of course possible to calculate sums, to perform shifts, multiplications and modulo divisions, as well as to calculate pseudorandom numbers with more complex instruction combinations. A good implementation of the presented method should rely on a set of building blocks which change a lot of key bytes and not just 32 bit. Simple xor instructions, as well as addition and subtraction are cryptographically weak, but the general code assembly method can be demonstrated best with these.

Instructions should alter the key quite frequently for not to offer the possibility to cryptoanalyze it by using a ciphertext codebook. When the method is used as pseudo random number generator, the result in ebx can be further processed. The internal state represented by the key data array is big enough for not to be directly or indirectly exposed.

An example for a much more cryptographically safe building block is a CRC32 implementation:

The function calculates a 32 bit CRC according to IEEE 802. The polynomial is: x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 +x + 1. X32 does of course not exist and the 1 only inverts the input data. Thus, the polynomial can be written as: $04C11DB7

```
  push ebp;               // ebp MUST never be really destroyed
  and  eax,127;        // perform an operation with four key bytes at a time using eax from the
                          previous instruction block
  add  ebp,eax;
  mov  eax,[ebp+0];        // load key[ebp+0] in AL and key[ebp+1] in the next upper byte of eax
                         and so on up to key[ebp+3]

  mov  esi,ebp;        // save ebp for later to alter the key
  pop  ebp;               // get original base of the key data array
  push ebp;               // restore stack frame
  push ebx;               // save ebx for later

  mov  ecx,32;               // counter for the loop
  xor  edx,edx;        // edx is used to clear the zero flag before the loop @rep1 command below
@rep1:rcl ebx,1;        // shift data in from ebx
  rcl  eax,1;               // use eax as CRC buffer
  jnc  @cnt1;               // CRC decision
  xor  eax,$04C11DB7;       // xor with IEEE 802 generator polynomial
@cnt1:add dl,1;        // clear Zero-Flag (will be rarely necessary)
  loop @rep1;
  pop  ebx;               // restore old ebx value. ebx keeps a running 32 bit result
  mov  ebp,esi;        // get the address of the previously selected key data bytes
  mov  [ebp+0],eax;        // alter the key
  xor  ebx,eax;        // alter ebx
  // here is the end of the CRC routine

  pop   ebp;         // exit the routine by restoring the original ebp
```

The presented building block only affects four key data bytes. Depending on the size of the key data array it should affect much more key bits for good attack security. It is very simple to extend the routine for satisfying this demand.

It is possible to add loops over one or more instruction blocks. This is usually performed by adding the 80386 loopne-command. The method spends more time on crunching instructions and that simply slows it down in order to make cryptoanalysis a time-consuming job. By altering ebp with every loop cycle, the key can influence the algorithm more often.

8192 bit keys are definitely too long. The increase in security with so many key bits is negligible compared to 256 bit. Uncrackable is simply uncrackable. In spite of this, the implementation of PMC in ciphers.de's BPP file encryption tool comes with this key size. Why not?

Attack security

Each instruction affects at least 32 bits of data and sometimes it alters the key.
If there are only 4 cryptographic instruction blocks and 16 of these blocks can be assembled chaotically one after the other, there exist 4^{16} = 4294967296 different possibilities for the actual encryption algorithm! If 128 instruction blocks were to be assembled, a choice of 4^{128} = $1{,}158*10^{77}$ combinations would result (standard 128 bit encryption yields a total of $3{,}403*10^{38}$).
It is important to note that this is without affecting execution time because there is the requirement for a well-shuffled key data array which must be guaranteed by conventional algorithms as well.

The Polymorphic Method features a substancially higher attack security than any conventional method. In order to calculate the total attack security, the number of code combinations must be multiplied by the number of key combinations. Key size may be 16 bytes = 128 bits; thus there exist 2^{128} = $3{,}403*10^{38}$ combinations for the key stored in the key data array. The two keyspaces multiplied yield $1{,}158*10^{77}$ * $3{,}403*10^{38}$ = $3{,}913*10^{115}$ possible key combinations for the Polymorphic Method.
In order to compare conventional cryptographic methods with the Polymorphic Method, the total keyspaces must be compared. As both methods are assumed to work on a 128 bit data key, this comparison is legal. Thus, the polymorphic method beats any conventional method by a factor of $3{,}913*10^{115}$ / $3{,}403*10^{38}$ = $1{,}150*10^{77}$ (!). This is more than the number of atoms on our planet!

The actual implementation in the cryptographic program „Best Possible Privacy“ uses 32 instructions and three bit of constant data per instruction. Thus, there are 32 ways to affect the algorithm multiplied by 8 possibilities for constant data => 256 = 2^8 variations
If the algorithm is limited to 1024 instruction blocks, there are $2^{(1024 * 8)}$ = 2^{8192} different code combinations possible and equally probable! The 256 byte keyspace further enhances attack security to yield 2^{8192} * 2^{2048} = 2^{10240}. Note that nearly 100% of the security comes from the compiler. The new method uses commonly known techniques but enhances them significantly.

Attacks and their likelyhood of success on the Polymorphic Method

Attacks are not algorithms, but instead just general approaches which must be reinvented for every new type of cipher.
It is generally assumed that The Opponent knows the design of the cipher and has virtually any amount of plaintext and corresponding ciphertext ("known plaintext"). It is further assumed that The Opponent has the real-time ability to obtain "defined plaintext" by enciphering messages at will and collecting the resulting ciphertext.

Exhaustive Search (Brute Force on the keys)

Try each possible key until the message deciphers properly. Try most-likely keys first.

A keyspace of at least 128 bits should be sufficient to prevent exhaustive search in the foreseeable future. The keying system for the Polymorphic Method is hard to implement with less than 256 bits and has usually a keyspace substancially beyond this value - around 2048 bits, not counting the key combinations for the instruction key which usually provide more than 99.9999999999% of the total security.

Chosen Key

Try various keys on known plaintext and compare the resulting ciphertext to the actual ciphertext, to try and build the correct key value.

As the key is more or less the algorithm itself, the task of an opponent is hopeless because the one-way polymorphic function comes in different shapes with each key, which is so big, that there is no possibility to isolate and work separately on some kind of table. A computer can only be as big as there are atoms on this planet.

Ciphertext-Only Codebook

Collect as many ciphertexts as possible and try to understand their contents through usage and relationships; then, when a ciphertext occurs, look it up. This treats the block cipher like a code, and is the

classic approach to code-breaking.

Just as some letters are more frequently used than others, words and phrases also have usage frequencies, as do blocks which contain plaintext. If the cipher block size is small (under 64 bytes), and if the ciphering key is not changed frequently, it may be possible to build a codebook of block values with their intended meanings.

Codebook attacks of any sort are ideally prevented by having a large number of block values, which implies a large block size. Once the block size is at least, say, 64 bytes, it can be expected that the amount of uniqueness in each block exceeds anyone's ability to collect and form a codebook.

Since the complexity of any sort of a codebook attack is related to block size only, doing "triple" anything will not affect increase this complexity. In particular, this means that Triple DES is no stronger than DES itself under this sort of attack, which is based on block size and not transformation complexity.

The Polymorphic Method is best implemented with a 1024 byte block size and the instruction sequence changing with every block. The method is further ideal for producing a seed for some random number generator which decouples the algorithm from the generation of the confusion sequence. Because a Polymorphic Method comes in different shapes with each key, any kind of codebook will contain mostly noise and will not be of great use.

Known Plaintext

Somehow "obtain" both the plaintext and the corresponding ciphertext for some large number of encipherings under one key.

With this kind of attack, one plaintext-ciphertext pair contains sufficient information to obtain the content of the key data array. In order to identify a key, both keys must be guessed using the Exhaustive Search method.
As both the input to the compiler as well as the keys are unknown, it is difficult to reveal the full internal state without revealing the underlying crypto system. The Polymorphic Method hides roughly three quarters of the internal state in the actual instruction code and that alone provides sufficient complexity. Note that a single known plaintext and ciphertext pair probably would identify a DES key!

Known-Plaintext Codebook

Collect as many ciphertexts and associated plaintext blocks as possible; then, when a ciphertext occurs, look it up.

Small block ciphers prevent codebook attacks by randomizing the plaintext (often with Cipher Block Chaining) so that the plaintext block values are distributed evenly across all possible block values.

Codebook attacks are ideally prevented by having a large number of block values, which implies a large block size. To prevent this attack for the future, a block size of 64 bytes is regarded as safe so the uniqueness it does contain assures that there will be too many different blocks to catalog. A 1024 byte block size and the use of a confusion sequence generator with at least 64 byte internal state makes it impossible to gain any ground on this kind of attack.
As the key is more or less the algorithm itself, the idea to create a table ends in logging noise.

Chosen Plaintext

Without knowing the key, arrange to cipher data at will and capture the associated ciphertext. Dynamically modify the data to reveal the key, or keyed values in the cipher.

The point here is not to decipher the associated ciphertext because the opponent is producing the original plaintext. If the opponents have chosen plaintext capabilities, they can probably also submit arbitrary ciphertext blocks for deciphering.

The weakness to be exploited here usually depends upon the ciphering system beyond the core cipher per se - a point with little internal state. As far as the Polymorphic Method is concerned, there is no static algorithm with some known weakness. Instead, there are a lot of possible weaknesses – each possible

keyed state. The Chosen Plaintext attack is not applicable here.

Chosen-Plaintext Codebook

Create as many ciphertexts and associated plaintext blocks as possible; then, when a ciphertext occurs, look it up.

This is much like the previous codebook attacks, now with the ability to fill the codebook at will and at electronic speeds. Again, the ability to do this depends upon the cipher having a relatively small block size and on a fixed cryptographic algorithm. This attack is again not applicable because it's simpler and equally efficient to try all possible keys.

Meet-in-the-Middle

With a multi-layered structure, given known-or defined-plaintext, search the top keyspace to find every possible result, and search the bottom keyspace to find every possible value.

With a two-level construct and a small block size, matches can be verified with a few subsequent known-plaintext/ciphertext pairs. Of course, three and more-level constructs can always be partitioned into two sections so a meet-in-the-middle attack can always be applied; this just may be pretty complex.

As each layer in a good crypto algorithm contains a huge amount of keyed state or „keyspace", the Polymorphic Method uses a large key and consequently adds a huge amount of unknown algorithm which multiplies with in the beginning unknown data keyspace to yield extraordinary complexity.

Key Bit Bias

Through extensive ciphering of fixed plaintext data under a variety of different keys, it may sometimes be possible to associate key bits with the statistical value of some ciphertext bits. This knowledge will break a conventional cipher quickly.

As different keys inevitably produce different cipher algorithms, statistics cannot help to link ciphertext with plaintext. There's simply a new independent variable in the game with the Polymorphic Method as each key state has some pretty unique weakness.

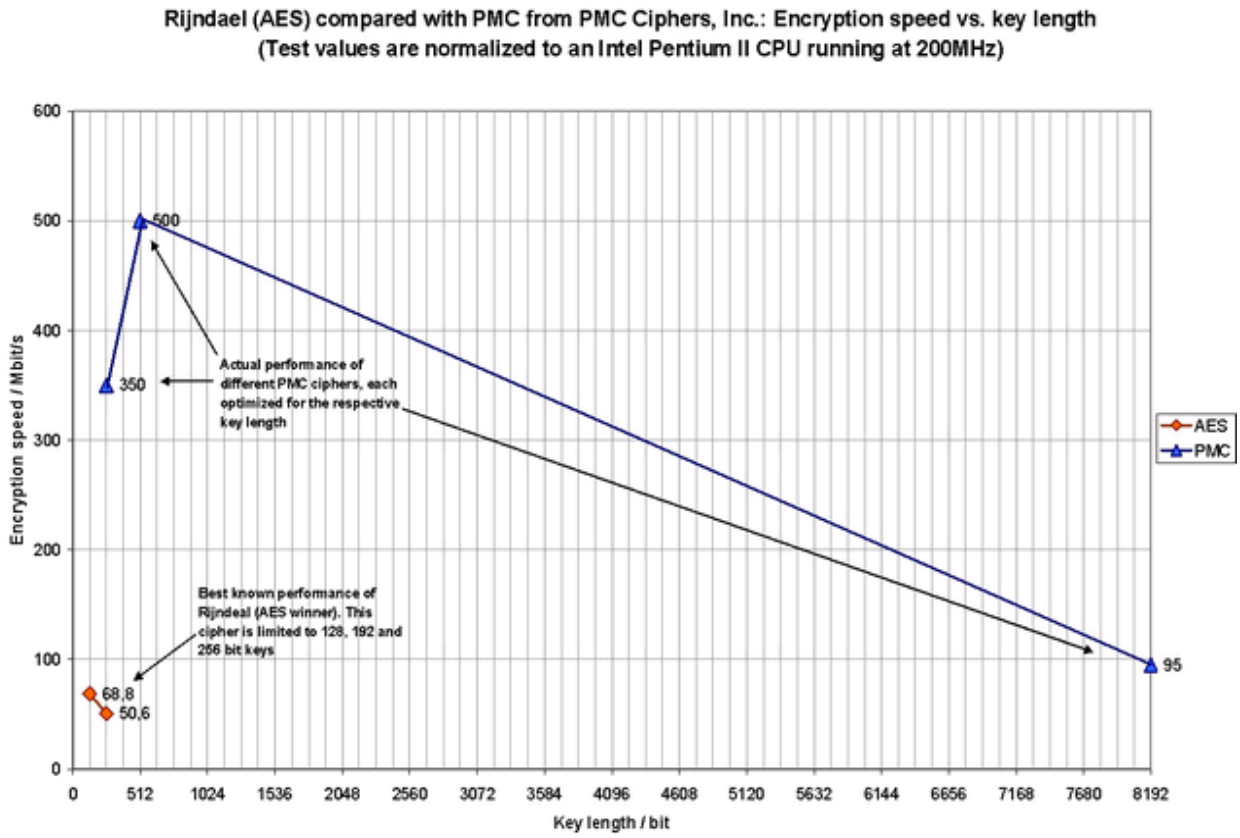
Differential Cryptanalysis

Exploit known properties of particular known substitution tables to effectively reduce the number of "rounds" in an iterated block cipher.

The original form of Differential Cryptanalysis mainly applies to iterated block ciphers with known tables, neither of which are present here. For an iterative cipher like DES, statistical unbalance can be found in known, fixed substitutions and that can be exploited to peer back into previous iteration steps.
For the Polymorphic Cipher Method, each different input value will actually select a different cipher, and this results in a completely variable transformation. It is hard and very inefficient to attack a transformation which changes it's structure completely whenever it is probed.

Summary

Except for the possibility to gain knowledge of the final state of the key data array in the basic configuration, there is nothing else to find out. There is no possibility to identify a key other than by searching exhaustively. The available keyspace is much greater than for any other cryptographic method. In order to compare the presented method with conventional methods, a conventional method has some data keyspace and only one possibility for the algorithm. The presented Polymorphic Method has the same data keyspace and an additional algorithm keyspace. All in all, the new method features a dramatic increase in security compared to common approaches.
It is worth imagining that cryptographically strong ciphers like DES, GOST, IDEA; Hashes, etc. are the building blocks of the Polymorphic Method. The weaknesses of each specific building block would vanish. The result would probably be a perfect cipher.



Speed

The original implementation of PMC in the TurboCrypt encryption tool is by far too slow when compared with the latest developments.
The latest variant with 512 bit key length is implemented in PMC Ciphers TurboCrypt. This crypto engine comes with an encryption speed of 500Mbit/s, which is approximately 10 times the speed of AES (Rijndael algorithm) operating with 256 bit keys!

Conclusion

For PMC, which was secret of state in 1999 in Germany, there exists no attack other than exhaustive search. There's no theoretical or practical way to to reconstruct keys from plaintext.
The presented method comes with a comparable number of „data keys" as conventional symmetric encryption methods. It adds a significant amount of possible and equally probable algorithmic keys, thus yielding substancially higher security and speed.
The 512 bit PMC crypto engine implemented in PMC Ciphers TurboCrypt uses the most probably fastest encryption algorithm in the world!

For more information: http://www.pmc-ciphers.com

-o-

256 and 512 bit PMC Block Encryption Algorithms

FACT SHEET

General Description

The 256 and 512 bit PMC Encryption Algorithms are designed for super-fast encryption/decryption and uncompromised security. Both Type 1 block ciphers have an integrated and selectable cipher feedback function.

Due to the Polymorphic nature of these ciphers, the actual encryption algorithm changes with the key. Perfect randomness, very high processing speed and immunity from every known attack result from this unique design.

Both crypto engines use the full internal state of 256/512 bit in a two-stage design with both stages compiled from the key during key-setup.

The second cipher stage is 100% intrinsically protected from Simple Power Attack (SPA), as well as from Differential Power Attack (DPA) making both ciphers the only encryption algorithms in the world which resist against every known attack.

The 256 bit PMC Block Cipher Engine is available as DLL and C++ source code. It integrates perfectly in existing and new designs live Voice-over-IP, Video-over-IP, VPN's, Network Routers, Fiber Optics Links, Satellite Channels, Disk Encryption, Encryption of the Operating System, File Encryption, License Management, DPA-proof Secure Smart Cards, etc.

The 256 bit PMC Block Cipher Engine encrypts / decrypts data 5 to 7 times faster than AES (Rijndael) while the 512 bit PMC Block Cipher Engine encrypts/decrypts data 10 times faster than AES in multi-block mode. Encrypting 512MByte and decrypting 512MByte on an AMD Athlon XP1800 processor takes only 1.6 seconds. This corresponds to an encryption speed of 5GBit/s.

Features

- 'Type 1' 256 and 512 bit block encryption
- Fully Polymorphic 2-stage design with both stages compiled from the key for optimum processing speed and data security
- DPA-proof Worker cipher stage (stage 2)
- Fastest known cipher, outperforming existing methods by factor 10
- Cipher Recompilation Mode capability for maximum protection of data streams with little entropy
- Easy integration in new and existing applications
- No known attack
- 256 and 512 bit Block PMC is the only available encryption algorithm for Secure Smart Cards
- 5GBit/s encryption speed using inexpensive general-purpose Microprocessors

Applications

- Replacement for unclassified ciphers like DES, Rijndael, and replacement for secret 'Type 1' ciphers with up to 512 key bits
- Fast VPNs with 10 times higher encryption speed
- Encryption of other server-to-server communication
- 1GBit Network Routers and (potentially) HAIPE devices
- High-speed backbones
- IP communication including encrypted Voice-over-IP, Video-on-demand, Webcasts, Video-over-IP
- Broadband Satellite Link Encryption
- Encryption of high-speed telecom links
- Broadband Military Applications
- Encryption of the Operating System for police cars, mobile military, etc.
- DPA-proof Secure Smart Cards
- Unbreakable Software License Management

Concept of PMC Encryption Algorithms

The concept of Polymorphic Encryption is based on the principle of compiled crypto code. A Crypto Compiler uses the passphrase to generate a large Pseudorandom Number Generator. This Compiled PRNG has the ability to alter the content of the Internal State in an unpredictable way.

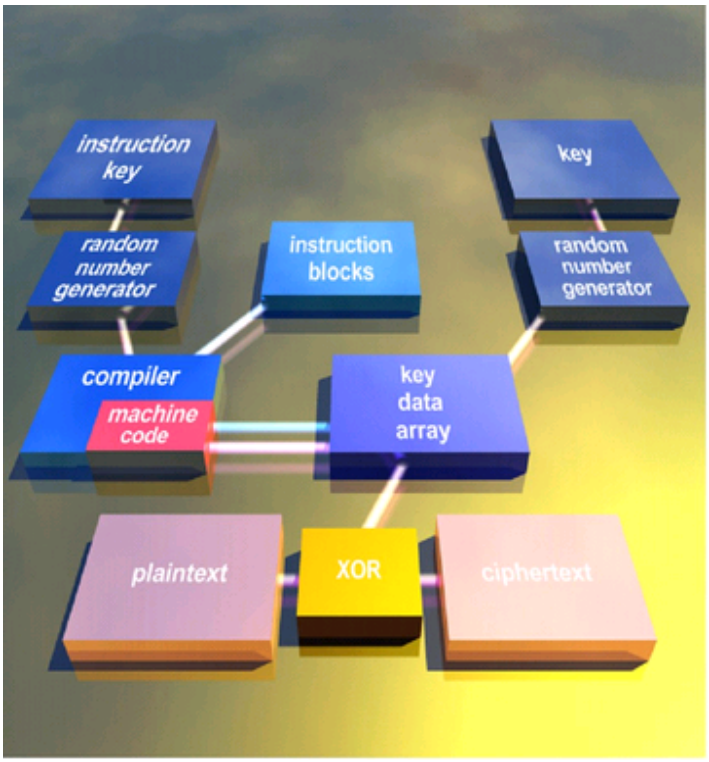


Fig. 1: Basic PMC structure

Unlike the structure shown in figure 1, which shows the simplest possible implementation of a Polymorphic Cipher, the 256 and 512 bit Block PMC Cipher Engines are two-stage implementations.

The content of the Internal State ("key data array") is used to bias an underlying fast Worker Cipher Stage. For the 256 and 512 bit Block PMC Cipher Engines, the Worker Cipher Stage is compiled as well from the passphrase.

Theoretical speed advantage of PMC

In contrast to common ciphers, which all come with the inherent speed limit $O(n^2)$ with n being the size of key k, the use of a crypto compiler has a positive effect on processing speed: There is only a linear relationship O(n) for the keysize n and the processing time.
The compiling process of the keystream generator can be generalized as block assembly with a constant number of key bits selecting the next block to be concatenated to the preceding ones. The processing time for that is O(n). The execution time for n primitive PRNGs is O(n), processing m plaintext bits with only a subset of the Internal State consumes O(m). Consequently the execution time of a Polymorphic Cipher is O(n)+ O(n)+ O(m).
The high encryption speed of Polymorphic Ciphers is unprecedented

Simplified Schematic

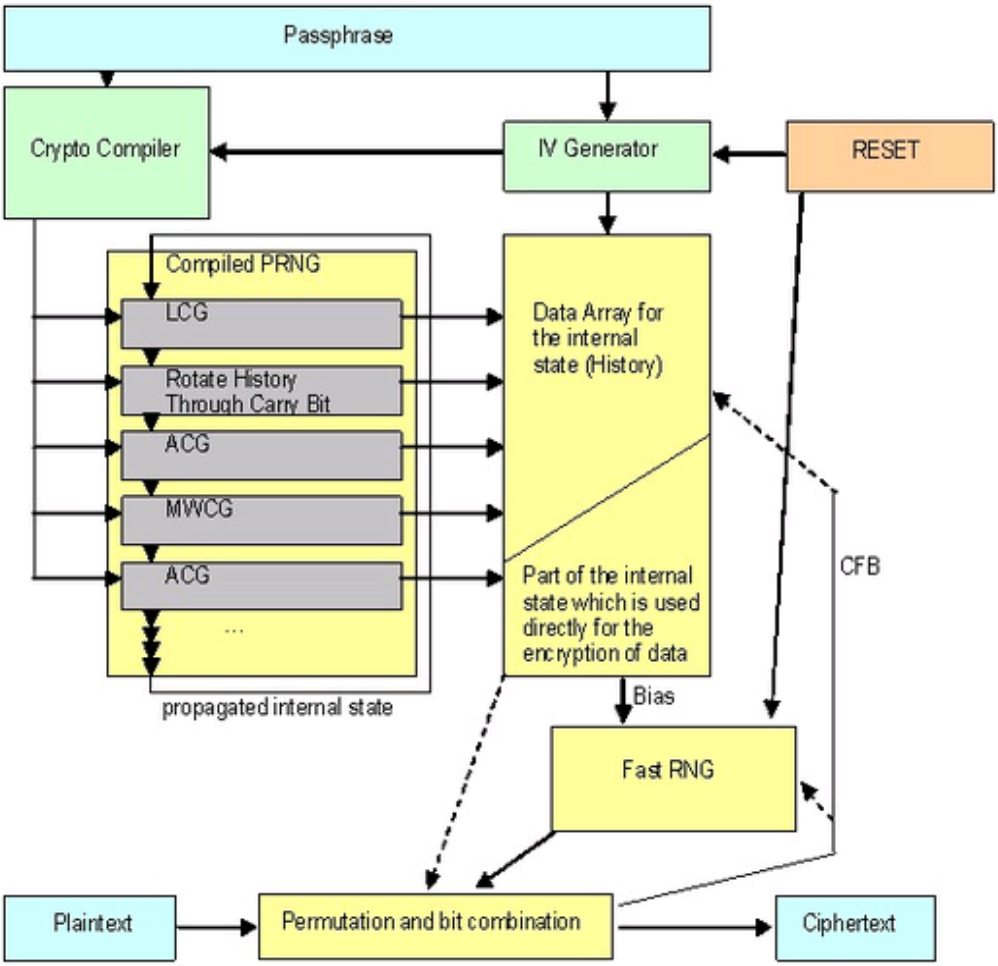


Fig. 2: Two-stage PMC structure

Encryption/Decryption speed normalized to AES test data

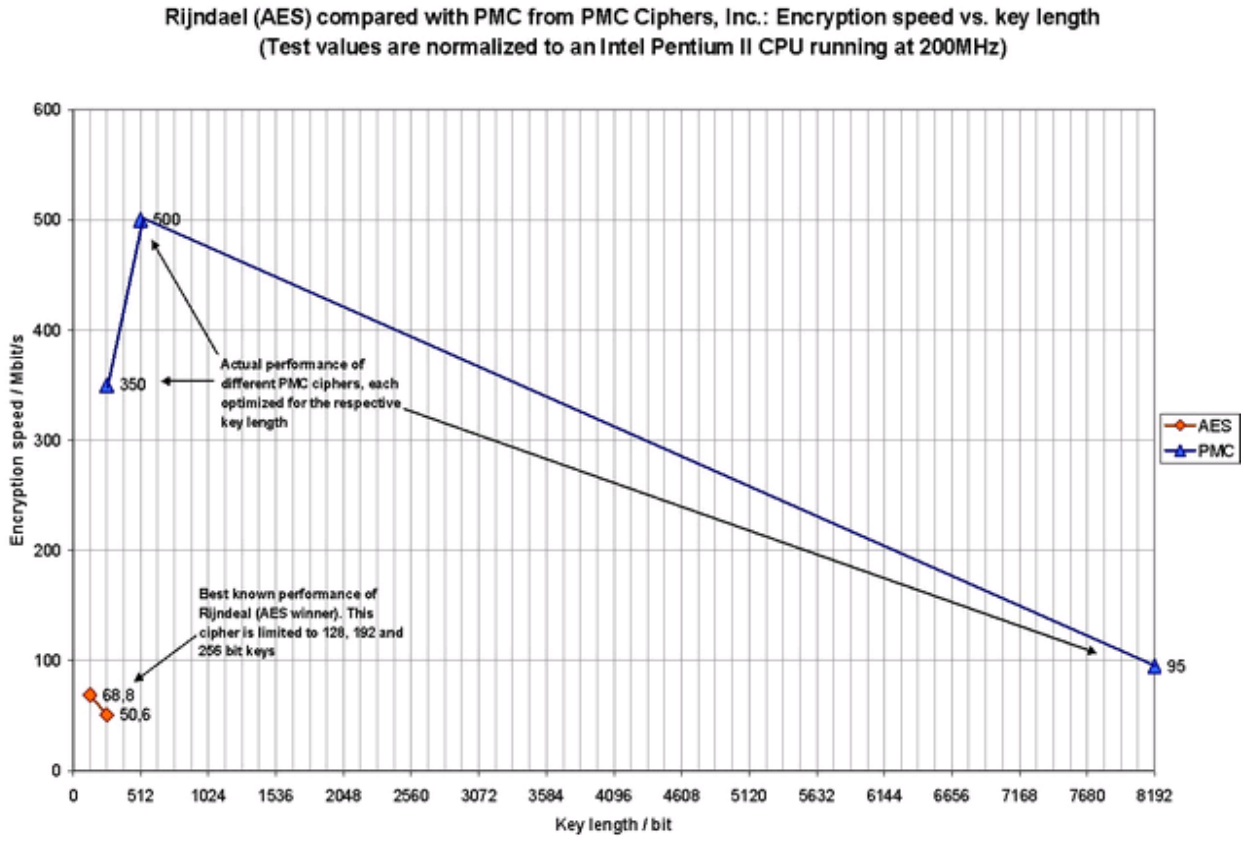


Fig. 3: Comparison of Two-stage PMC block ciphers with different key length and AES (Rijndael)

-o-

Diehard[1] Test on the 512 bit Polymorphic Block Cipher implemented in TurboCrypt - Results

Technical Paper

First published: August 2003

For the latest information, please see http://www.pmc-ciphers.com

Introduction

The novel 512 bit Polymorphic Block Cipher Engine of PMC Ciphers, Inc. used in the product TurboCrypt is subject to extensive randomness test.
A constant stream of zero bits was encrypted by the TurboCrypt V6.1 Control Panel using the following password:
1111111111111111111

The 128 MB volume file was split into two parts of 64MB size each. Only the second part was used for the test because the leading 512 bytes of the first part contain FAT16 boot sector data as plaintext.

The well-known and frequently applied test suite for randomness is “Diehard” by George Marsaglia. It is a battery of tests which consists of:

BIRTHDAY SPACINGS TEST, OVERLAPPING 5-PERMUTATION TEST, BINARY RANK TEST for 31x31 matrices, BINARY RANK TEST for 32x32 matrices, BINARY RANK TEST for 6x8 matrices, BITSTREAM TEST, Overlapping-Pairs-Sparse-Occupancy (OPSO), Overlapping-Quadruples-Sparse-Occupancy (OQSO), DNA, COUNT-THE-1's TEST, COUNT-THE-1's TEST for specific bytes, PARKING LOT TEST, MINIMUM DISTANCE TEST, 3DSPHERES TEST, SQEEZE TEST, OVERLAPPING SUMS TEST, RUNS TEST and the CRAPS TEST.

Test results

It is important to note that most of the tests in DIEHARD return a p-value, which should be uniform on [0,1] if the input file contains truly independent random bits.
George Marsaglia writes that those p-values are obtained by p=F(X), where F is the assumed distribution of the sample random variable X---often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus you should not be surprised with occasional p-values near 0 or 1, such as .0012 or .9983. When a bit stream really FAILS BIG, you will get p's of 0 or 1 to six or more places. By all means, do not, as a Statistician might, think that a p < .025 or p> .975 means that the RNG has "failed the test at the .05 level". Such p's happen among the hundreds that DIEHARD produces, even with good RNG's. So keep in mind that " p happens".

Here’s the summary of results:

```
BIRTHDAY SPACINGS TEST, M= 512 N=2**24 LAMBDA=  2.0000
           Results for c:\v.vol
                    For a sample of size 500:     mean
           c:\v.vol        using bits  1 to 24   2.018
  duplicate       number       number
  spacings       observed     expected
        0          77.       67.668
        1         132.      135.335
        2         132.      135.335
        3          69.       90.224
        4          56.       45.112
        5          24.       18.045
  6 to INF         10.        8.282
 Chisquare with  6 d.o.f. =    11.39 p-value=  .923059
  ::::::::::::::::::::::::::::::::::::::::
                    For a sample of size 500:     mean
           c:\v.vol        using bits  2 to 25   2.036
  duplicate       number       number
  spacings       observed     expected
        0          69.       67.668
        1         138.      135.335
        2         130.      135.335
        3          82.       90.224
        4          51.       45.112
        5          17.       18.045
  6 to INF         13.        8.282
 Chisquare with  6 d.o.f. =     4.56 p-value=  .398066
  ::::::::::::::::::::::::::::::::::::::::
                    For a sample of size 500:     mean
           c:\v.vol        using bits  3 to 26   2.074
  duplicate       number       number
  spacings       observed     expected
        0          62.       67.668
        1         128.      135.335
        2         144.      135.335
        3          81.       90.224
        4          56.       45.112
        5          21.       18.045
  6 to INF          8.        8.282
 Chisquare with  6 d.o.f. =     5.49 p-value=  .517521
  ::::::::::::::::::::::::::::::::::::::::
                    For a sample of size 500:     mean
           c:\v.vol        using bits  4 to 27   2.044
  duplicate       number       number
  spacings       observed     expected
        0          70.       67.668
```

```
       1         114.      135.335
       2         150.      135.335
       3          91.       90.224
       4          47.       45.112
       5          22.       18.045
 6 to INF          6.        8.282
Chisquare with  6 d.o.f. =     6.61 p-value=  .642005
 ::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
         c:\v.vol        using bits  5 to 28   2.074
 duplicate       number       number
 spacings       observed     expected
       0          62.       67.668
       1         126.      135.335
       2         140.      135.335
       3          94.       90.224
       4          51.       45.112
       5          20.       18.045
 6 to INF          7.        8.282
Chisquare with  6 d.o.f. =     2.62 p-value=  .144772
 ::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
         c:\v.vol        using bits  6 to 29   2.038
 duplicate       number       number
 spacings       observed     expected
       0          62.       67.668
       1         143.      135.335
       2         124.      135.335
       3          89.       90.224
       4          56.       45.112
       5          20.       18.045
 6 to INF          6.        8.282
Chisquare with  6 d.o.f. =     5.34 p-value=  .499413
 ::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
         c:\v.vol        using bits  7 to 30   2.016
 duplicate       number       number
 spacings       observed     expected
       0          66.       67.668
       1         124.      135.335
       2         146.      135.335
       3          94.       90.224
       4          51.       45.112
       5          11.       18.045
 6 to INF          8.        8.282
Chisquare with  6 d.o.f. =     5.52 p-value=  .520645
 ::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
         c:\v.vol        using bits  8 to 31   2.018
 duplicate       number       number
 spacings       observed     expected
       0          57.       67.668
       1         147.      135.335
       2         137.      135.335
       3          86.       90.224
       4          47.       45.112
       5          18.       18.045
 6 to INF          8.        8.282
Chisquare with  6 d.o.f. =     2.99 p-value=  .190407
 ::::::::::::::::::::::::::::::::::::::::
                For a sample of size 500:     mean
         c:\v.vol        using bits  9 to 32   1.946
 duplicate       number       number
 spacings       observed     expected
       0          70.       67.668
       1         144.      135.335
       2         133.      135.335
       3          81.       90.224
       4          48.       45.112
       5          17.       18.045
 6 to INF          7.        8.282
Chisquare with  6 d.o.f. =     2.06 p-value=  .086105
 ::::::::::::::::::::::::::::::::::::::::
  The 9 p-values were
      .923059   .398066   .517521   .642005   .144772
      .499413   .520645   .190407   .086105
 A KSTEST for the 9 p-values yields  .291151

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    ::            THE OVERLAPPING 5-PERMUTATION TEST                  ::
    :: This is the OPERM5 test.  It looks at a sequence of one mill- ::
    :: ion 32-bit random integers.  Each set of five consecutive     ::
    :: integers can be in one of 120 states, for the 5! possible or- ::
    :: derings of five numbers.  Thus the 5th, 6th, 7th,...numbers   ::
    :: each provide a state. As many thousands of state transitions  ::
    :: are observed,  cumulative counts are made of the number of    ::
    :: occurences of each state.  Then the quadratic form in the     ::
    :: weak inverse of the 120x120 covariance matrix yields a test   ::
    :: equivalent to the likelihood ratio test that the 120 cell     ::
    :: counts came from the specified (asymptotically) normal dis-   ::
    :: tribution with the specified 120x120 covariance matrix (with  ::
    :: rank 99).  This version uses 1,000,000 integers, twice.       ::
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
         OPERM5 test for file c:\v.vol
```

```
    For a sample of 1,000,000 consecutive 5-tuples,
 chisquare for 99 degrees of freedom= 59.510; p-value= .000585
          OPERM5 test for file c:\v.vol
    For a sample of 1,000,000 consecutive 5-tuples,
 chisquare for 99 degrees of freedom=100.597; p-value= .563675
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    :: This is the BINARY RANK TEST for 31x31 matrices. The leftmost ::
    :: 31 bits of 31 random integers from the test sequence are used ::
    :: to form a 31x31 binary matrix over the field {0,1}. The rank  ::
    :: is determined. That rank can be from 0 to 31, but ranks< 28   ::
    :: are rare, and their counts are pooled with those for rank 28. ::
    :: Ranks are found for 40,000 such random matrices and a chisqua-::
    :: re test is performed on counts for ranks 31,30,29 and <=28.   ::
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
   Binary rank test for c:\v.vol
        Rank test for 31x31 binary matrices:
       rows from leftmost 31 bits of each 32-bit integer
     rank   observed  expected (o-e)^2/e  sum
       28       210     211.4   .009511     .010
       29      5202    5134.0   .900389     .910
       30     23014   23103.0   .343216    1.253
       31     11574   11551.5   .043730    1.297
  chisquare= 1.297 for 3 d. of f.; p-value= .397513
-------------------------------------------------------------
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    :: This is the BINARY RANK TEST for 32x32 matrices. A random 32x ::
    :: 32 binary matrix is formed, each row a 32-bit random integer. ::
    :: The rank is determined. That rank can be from 0 to 32, ranks  ::
    :: less than 29 are rare, and their counts are pooled with those ::
    :: for rank 29.  Ranks are found for 40,000 such random matrices ::
    :: and a chisquare test is performed on counts for ranks  32,31, ::
    :: 30 and <=29.                                                  ::
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
   Binary rank test for c:\v.vol
        Rank test for 32x32 binary matrices:
       rows from leftmost 32 bits of each 32-bit integer
     rank   observed  expected (o-e)^2/e  sum
       29       201     211.4   .513367     .513
       30      5122    5134.0   .028096     .541
       31     23076   23103.0   .031664     .573
       32     11601   11551.5   .211906     .785
  chisquare=  .785 for 3 d. of f.; p-value= .334760
-------------------------------------------------------------

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    :: This is the BINARY RANK TEST for 6x8 matrices.  From each of  ::
    :: six random 32-bit integers from the generator under test, a   ::
    :: specified byte is chosen, and the resulting six bytes form a  ::
    :: 6x8 binary matrix whose rank is determined.  That rank can be ::
    :: from 0 to 6, but ranks 0,1,2,3 are rare; their counts are     ::
    :: pooled with those for rank 4. Ranks are found for 100,000     ::
    :: random matrices, and a chi-square test is performed on        ::
    :: counts for ranks 6,5 and <=4.                                 ::
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
        Binary Rank Test for c:\v.vol
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits  1 to  8
                    OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4          976       944.3       1.064       1.064
         r =5        21988     21743.9       2.740       3.804
         r =6        77036     77311.8        .984       4.788
                       p=1-exp(-SUM/2)= .90875
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits  2 to  9
                    OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4          953       944.3        .080        .080
         r =5        21899     21743.9       1.106       1.186
         r =6        77148     77311.8        .347       1.534
                       p=1-exp(-SUM/2)= .53548
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits  3 to 10
                    OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4          939       944.3        .030        .030
         r =5        21652     21743.9        .388        .418
         r =6        77409     77311.8        .122        .540
                       p=1-exp(-SUM/2)= .23676
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits  4 to 11
                    OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4          941       944.3        .012        .012
         r =5        21648     21743.9        .423        .435
         r =6        77411     77311.8        .127        .562
                       p=1-exp(-SUM/2)= .24489
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits  5 to 12
                    OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4          879       944.3       4.516       4.516
         r =5        21814     21743.9        .226       4.742
         r =6        77307     77311.8        .000       4.742
```

```
                  p=1-exp(-SUM/2)= .90662
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits  6 to 13
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          939       944.3        .030        .030
     r =5        21903     21743.9       1.164       1.194
     r =6        77158     77311.8        .306       1.500
                  p=1-exp(-SUM/2)= .52760
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits  7 to 14
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          971       944.3        .755        .755
     r =5        21827     21743.9        .318       1.072
     r =6        77202     77311.8        .156       1.228
                  p=1-exp(-SUM/2)= .45893
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits  8 to 15
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4         1001       944.3       3.404       3.404
     r =5        21695     21743.9        .110       3.514
     r =6        77304     77311.8        .001       3.515
                  p=1-exp(-SUM/2)= .82754
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits  9 to 16
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          999       944.3       3.168       3.168
     r =5        21488     21743.9       3.012       6.180
     r =6        77513     77311.8        .524       6.704
                  p=1-exp(-SUM/2)= .96498
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits 10 to 17
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          873       944.3       5.384       5.384
     r =5        21982     21743.9       2.607       7.991
     r =6        77145     77311.8        .360       8.351
                  p=1-exp(-SUM/2)= .98463
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits 11 to 18
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          932       944.3        .160        .160
     r =5        21634     21743.9        .555        .716
     r =6        77434     77311.8        .193        .909
                  p=1-exp(-SUM/2)= .36519
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits 12 to 19
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          895       944.3       2.574       2.574
     r =5        21731     21743.9        .008       2.582
     r =6        77374     77311.8        .050       2.632
                  p=1-exp(-SUM/2)= .73175
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits 13 to 20
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          895       944.3       2.574       2.574
     r =5        21575     21743.9       1.312       3.886
     r =6        77530     77311.8        .616       4.502
                  p=1-exp(-SUM/2)= .89469
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits 14 to 21
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          956       944.3        .145        .145
     r =5        21608     21743.9        .849        .994
     r =6        77436     77311.8        .200       1.194
                  p=1-exp(-SUM/2)= .44949
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits 15 to 22
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          969       944.3        .646        .646
     r =5        21580     21743.9       1.235       1.881
     r =6        77451     77311.8        .251       2.132
                  p=1-exp(-SUM/2)= .65563
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits 16 to 23
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          950       944.3        .034        .034
     r =5        21723     21743.9        .020        .054
     r =6        77327     77311.8        .003        .057
                  p=1-exp(-SUM/2)= .02832
   Rank of a 6x8 binary matrix,
rows formed from eight bits of the RNG c:\v.vol
b-rank test for bits 17 to 24
                OBSERVED   EXPECTED     (O-E)^2/E      SUM
     r<=4          982       944.3       1.505       1.505
     r =5        21826     21743.9        .310       1.815
     r =6        77192     77311.8        .186       2.001
```

```
                    p=1-exp(-SUM/2)= .63224
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits 18 to 25
                 OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4        948       944.3        .014        .014
         r =5      21583     21743.9       1.191       1.205
         r =6      77469     77311.8        .320       1.525
                    p=1-exp(-SUM/2)= .53344
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits 19 to 26
                 OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4        948       944.3        .014        .014
         r =5      21790     21743.9        .098        .112
         r =6      77262     77311.8        .032        .144
                    p=1-exp(-SUM/2)= .06961
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits 20 to 27
                 OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4        948       944.3        .014        .014
         r =5      21658     21743.9        .339        .354
         r =6      77394     77311.8        .087        .441
                    p=1-exp(-SUM/2)= .19797
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits 21 to 28
                 OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4        934       944.3        .112        .112
         r =5      21495     21743.9       2.849       2.962
         r =6      77571     77311.8        .869       3.830
                    p=1-exp(-SUM/2)= .85269
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits 22 to 29
                 OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4        955       944.3        .121        .121
         r =5      21777     21743.9        .050        .172
         r =6      77268     77311.8        .025        .196
                    p=1-exp(-SUM/2)= .09354
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits 23 to 30
                 OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4        938       944.3        .042        .042
         r =5      21904     21743.9       1.179       1.221
         r =6      77158     77311.8        .306       1.527
                    p=1-exp(-SUM/2)= .53393
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits 24 to 31
                 OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4        999       944.3       3.168       3.168
         r =5      21630     21743.9        .597       3.765
         r =6      77371     77311.8        .045       3.810
                    p=1-exp(-SUM/2)= .85121
       Rank of a 6x8 binary matrix,
    rows formed from eight bits of the RNG c:\v.vol
    b-rank test for bits 25 to 32
                 OBSERVED   EXPECTED     (O-E)^2/E      SUM
         r<=4        935       944.3        .092        .092
         r =5      21623     21743.9        .672        .764
         r =6      77442     77311.8        .219        .983
                    p=1-exp(-SUM/2)= .38832
  TEST SUMMARY, 25 tests on 100,000 random 6x8 matrices
 These should be 25 uniform [0,1] random variables:
    .908749     .535485     .236762     .244888     .906616
    .527603     .458928     .827535     .964980     .984631
    .365188     .731750     .894693     .449492     .655628
    .028325     .632242     .533440     .069613     .197974
    .852694     .093541     .533929     .851207     .388322
  brank test summary for c:\v.vol
      The KS test for those 25 supposed UNI's yields
                   KS p-value= .415163

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    ::                     THE BITSTREAM TEST                         ::
    :: The file under test is viewed as a stream of bits. Call them  ::
    :: b1,b2,... .  Consider an alphabet with two "letters", 0 and 1 ::
    :: and think of the stream of bits as a succession of 20-letter  ::
    :: "words", overlapping.  Thus the first word is b1b2...b20, the ::
    :: second is b2b3...b21, and so on.  The bitstream test counts   ::
    :: the number of missing 20-letter (20-bit) words in a string of ::
    :: 2^21 overlapping 20-letter words.  There are 2^20 possible 20 ::
    :: letter words.  For a truly random string of 2^21+19 bits, the ::
    :: number of missing words j should be (very close to) normally  ::
    :: distributed with mean 141,909 and sigma 428.  Thus            ::
    ::  (j-141909)/428 should be a standard normal variate (z score) ::
    :: that leads to a uniform [0,1) p value.  The test is repeated  ::
    :: twenty times.                                                 ::
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 THE OVERLAPPING 20-tuples BITSTREAM  TEST, 20 BITS PER WORD, N words
   This test uses N=2^21 and samples the bitstream 20 times.
```

```
  No. missing words should average  141909. with sigma=428.
-----------------------------------------------------------
 tst no  1:  142685 missing words,    1.81 sigmas from mean, p-value= .96503
 tst no  2:  141519 missing words,    -.91 sigmas from mean, p-value= .18089
 tst no  3:  142341 missing words,    1.01 sigmas from mean, p-value= .84341
 tst no  4:  142050 missing words,     .33 sigmas from mean, p-value= .62880
 tst no  5:  142099 missing words,     .44 sigmas from mean, p-value= .67117
 tst no  6:  141486 missing words,    -.99 sigmas from mean, p-value= .16131
 tst no  7:  142352 missing words,    1.03 sigmas from mean, p-value= .84950
 tst no  8:  142005 missing words,     .22 sigmas from mean, p-value= .58844
 tst no  9:  142566 missing words,    1.53 sigmas from mean, p-value= .93752
 tst no 10:  141598 missing words,    -.73 sigmas from mean, p-value= .23349
 tst no 11:  142881 missing words,    2.27 sigmas from mean, p-value= .98840
 tst no 12:  141064 missing words,   -1.98 sigmas from mean, p-value= .02413
 tst no 13:  142201 missing words,     .68 sigmas from mean, p-value= .75221
 tst no 14:  142285 missing words,     .88 sigmas from mean, p-value= .80996
 tst no 15:  141928 missing words,     .04 sigmas from mean, p-value= .51740
 tst no 16:  142442 missing words,    1.24 sigmas from mean, p-value= .89335
 tst no 17:  141861 missing words,    -.11 sigmas from mean, p-value= .45505
 tst no 18:  142324 missing words,     .97 sigmas from mean, p-value= .83369
 tst no 19:  140772 missing words,   -2.66 sigmas from mean, p-value= .00394
 tst no 20:  142179 missing words,     .63 sigmas from mean, p-value= .73568

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::             The tests OPSO, OQSO and DNA                       ::
     ::         OPSO means Overlapping-Pairs-Sparse-Occupancy           ::
     :: The OPSO test considers 2-letter words from an alphabet of      ::
     :: 1024 letters.  Each letter is determined by a specified ten     ::
     :: bits from a 32-bit integer in the sequence to be tested. OPSO   ::
     :: generates  2^21 (overlapping) 2-letter words  (from 2^21+1      ::
     :: "keystrokes")  and counts the number of missing words---that    ::
     :: is 2-letter words which do not appear in the entire sequence.   ::
     :: That count should be very close to normally distributed with    ::
     :: mean 141,909, sigma 290. Thus (missingwrds-141909)/290 should   ::
     :: be a standard normal variable. The OPSO test takes 32 bits at   ::
     :: a time from the test file and uses a designated set of ten      ::
     :: consecutive bits. It then restarts the file for the next de-    ::
     :: signated 10 bits, and so on.                                    ::
     ::                                                                 ::
     ::     OQSO means Overlapping-Quadruples-Sparse-Occupancy          ::
     ::   The test OQSO is similar, except that it considers 4-letter   ::
     :: words from an alphabet of 32 letters, each letter determined    ::
     :: by a designated string of 5 consecutive bits from the test      ::
     :: file, elements of which are assumed 32-bit random integers.     ::
     :: The mean number of missing words in a sequence of 2^21 four-    ::
     :: letter words,  (2^21+3 "keystrokes"), is again 141909, with     ::
     :: sigma = 295.  The mean is based on theory; sigma comes from     ::
     :: extensive simulation.                                           ::
     ::                                                                 ::
     ::    The DNA test considers an alphabet of 4 letters::  C,G,A,T,  ::
     :: determined by two designated bits in the sequence of random     ::
     :: integers being tested.  It considers 10-letter words, so that   ::
     :: as in OPSO and OQSO, there are 2^20 possible words, and the     ::
     :: mean number of missing words from a string of 2^21  (over-      ::
     :: lapping)  10-letter  words (2^21+9 "keystrokes") is 141909.     ::
     :: The standard deviation sigma=339 was determined as for OQSO     ::
     :: by simulation.  (Sigma for OPSO, 290, is the true value (to     ::
     :: three places), not determined by simulation.                    ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 OPSO test for generator c:\v.vol
  Output: No. missing words (mw), equiv normal variate (z), p-value (p)
                                                          mw     z     p
    OPSO for c:\v.vol       using bits 23 to 32       142381  1.626  .9481
    OPSO for c:\v.vol       using bits 22 to 31       141840  -.239  .4055
    OPSO for c:\v.vol       using bits 21 to 30       142659  2.585  .9951
    OPSO for c:\v.vol       using bits 20 to 29       142082   .595  .7242
    OPSO for c:\v.vol       using bits 19 to 28       141451 -1.580  .0570
    OPSO for c:\v.vol       using bits 18 to 27       141755  -.532  .2973
    OPSO for c:\v.vol       using bits 17 to 26       141860  -.170  .4325
    OPSO for c:\v.vol       using bits 16 to 25       141956   .161  .5639
    OPSO for c:\v.vol       using bits 15 to 24       141917   .026  .5106
    OPSO for c:\v.vol       using bits 14 to 23       141429 -1.656  .0488
    OPSO for c:\v.vol       using bits 13 to 22       142147   .820  .7938
    OPSO for c:\v.vol       using bits 12 to 21       142093   .633  .7367
    OPSO for c:\v.vol       using bits 11 to 20       142023   .392  .6525
    OPSO for c:\v.vol       using bits 10 to 19       141820  -.308  .3790
    OPSO for c:\v.vol       using bits  9 to 18       142255  1.192  .8834
    OPSO for c:\v.vol       using bits  8 to 17       141797  -.387  .3493
    OPSO for c:\v.vol       using bits  7 to 16       141890  -.067  .4734
    OPSO for c:\v.vol       using bits  6 to 15       141516 -1.356  .0875
    OPSO for c:\v.vol       using bits  5 to 14       142096   .644  .7401
    OPSO for c:\v.vol       using bits  4 to 13       141996   .299  .6175
    OPSO for c:\v.vol       using bits  3 to 12       142179   .930  .8238
    OPSO for c:\v.vol       using bits  2 to 11       141667  -.836  .2017
    OPSO for c:\v.vol       using bits  1 to 10       141556 -1.218  .1115
 OQSO test for generator c:\v.vol
  Output: No. missing words (mw), equiv normal variate (z), p-value (p)
                                                          mw     z     p
    OQSO for c:\v.vol       using bits 28 to 32       142142   .789  .7849
    OQSO for c:\v.vol       using bits 27 to 31       141906  -.011  .4955
    OQSO for c:\v.vol       using bits 26 to 30       141950   .138  .5548
    OQSO for c:\v.vol       using bits 25 to 29       142367  1.551  .9396
    OQSO for c:\v.vol       using bits 24 to 28       142161   .853  .8032
    OQSO for c:\v.vol       using bits 23 to 27       142001   .311  .6220
```

```
    OQSO for c:\v.vol        using bits 22 to 26        142116   .701  .7582
    OQSO for c:\v.vol        using bits 21 to 25        141744  -.560  .2876
    OQSO for c:\v.vol        using bits 20 to 24        141965   .189  .5748
    OQSO for c:\v.vol        using bits 19 to 23        142023   .385  .6500
    OQSO for c:\v.vol        using bits 18 to 22        141626  -.960  .1684
    OQSO for c:\v.vol        using bits 17 to 21        141994   .287  .6130
    OQSO for c:\v.vol        using bits 16 to 20        141845  -.218  .4137
    OQSO for c:\v.vol        using bits 15 to 19        141882  -.093  .4631
    OQSO for c:\v.vol        using bits 14 to 18        142045   .460  .6772
    OQSO for c:\v.vol        using bits 13 to 17        141963   .182  .5722
    OQSO for c:\v.vol        using bits 12 to 16        142011   .345  .6348
    OQSO for c:\v.vol        using bits 11 to 15        141425 -1.642  .0503
    OQSO for c:\v.vol        using bits 10 to 14        141870  -.133  .4470
    OQSO for c:\v.vol        using bits  9 to 13        141938   .097  .5387
    OQSO for c:\v.vol        using bits  8 to 12        142121   .718  .7635
    OQSO for c:\v.vol        using bits  7 to 11        142052   .484  .6857
    OQSO for c:\v.vol        using bits  6 to 10        142134   .762  .7769
    OQSO for c:\v.vol        using bits  5 to  9        141851  -.198  .4216
    OQSO for c:\v.vol        using bits  4 to  8        141370 -1.828  .0338
    OQSO for c:\v.vol        using bits  3 to  7        141843  -.225  .4111
    OQSO for c:\v.vol        using bits  2 to  6        141905  -.015  .4941
    OQSO for c:\v.vol        using bits  1 to  5        141622  -.974  .1650
  DNA test for generator c:\v.vol
  Output: No. missing words (mw), equiv normal variate (z), p-value (p)
                                                          mw     z     p
     DNA for c:\v.vol        using bits 31 to 32        141944   .102  .5407
     DNA for c:\v.vol        using bits 30 to 31        141725  -.544  .2933
     DNA for c:\v.vol        using bits 29 to 30        141667  -.715  .2374
     DNA for c:\v.vol        using bits 28 to 29        141887  -.066  .4737
     DNA for c:\v.vol        using bits 27 to 28        141957   .141  .5559
     DNA for c:\v.vol        using bits 26 to 27        141889  -.060  .4761
     DNA for c:\v.vol        using bits 25 to 26        141469 -1.299  .0970
     DNA for c:\v.vol        using bits 24 to 25        141723  -.550  .2913
     DNA for c:\v.vol        using bits 23 to 24        141964   .161  .5641
     DNA for c:\v.vol        using bits 22 to 23        141904  -.016  .4937
     DNA for c:\v.vol        using bits 21 to 22        141747  -.479  .3160
     DNA for c:\v.vol        using bits 20 to 21        141667  -.715  .2374
     DNA for c:\v.vol        using bits 19 to 20        142048   .409  .6588
     DNA for c:\v.vol        using bits 18 to 19        141635  -.809  .2092
     DNA for c:\v.vol        using bits 17 to 18        141930   .061  .5243
     DNA for c:\v.vol        using bits 16 to 17        141281 -1.853  .0319
     DNA for c:\v.vol        using bits 15 to 16        142149   .707  .7602
     DNA for c:\v.vol        using bits 14 to 15        142398  1.442  .9253
     DNA for c:\v.vol        using bits 13 to 14        141346 -1.662  .0483
     DNA for c:\v.vol        using bits 12 to 13        141979   .206  .5814
     DNA for c:\v.vol        using bits 11 to 12        141876  -.098  .4608
     DNA for c:\v.vol        using bits 10 to 11        141960   .149  .5594
     DNA for c:\v.vol        using bits  9 to 10        142269  1.061  .8557
     DNA for c:\v.vol        using bits  8 to  9        142060   .444  .6716
     DNA for c:\v.vol        using bits  7 to  8        141438 -1.390  .0822
     DNA for c:\v.vol        using bits  6 to  7        142002   .273  .6077
     DNA for c:\v.vol        using bits  5 to  6        141818  -.269  .3938
     DNA for c:\v.vol        using bits  4 to  5        142038   .380  .6479
     DNA for c:\v.vol        using bits  3 to  4        141785  -.367  .3569
     DNA for c:\v.vol        using bits  2 to  3        141657  -.744  .2283
     DNA for c:\v.vol        using bits  1 to  2        142175   .784  .7834

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::     This is the COUNT-THE-1's TEST on a stream of bytes.      ::
     :: Consider the file under test as a stream of bytes (four per   ::
     :: 32 bit integer).  Each byte can contain from 0 to 8 1's,      ::
     :: with probabilities 1,8,28,56,70,56,28,8,1 over 256.  Now let  ::
     :: the stream of bytes provide a string of overlapping  5-letter ::
     :: words, each "letter" taking values A,B,C,D,E. The letters are ::
     :: determined by the number of 1's in a byte::  0,1,or 2 yield A,::
     :: 3 yields B, 4 yields C, 5 yields D and 6,7 or 8 yield E. Thus ::
     :: we have a monkey at a typewriter hitting five keys with vari- ::
     :: ous probabilities (37,56,70,56,37 over 256).  There are 5^5   ::
     :: possible 5-letter words, and from a string of 256,000 (over-  ::
     :: lapping) 5-letter words, counts are made on the frequencies   ::
     :: for each word.   The quadratic form in the weak inverse of    ::
     :: the covariance matrix of the cell counts provides a chisquare ::
     :: test::  Q5-Q4, the difference of the naive Pearson sums of    ::
     :: (OBS-EXP)^2/EXP on counts for 5- and 4-letter cell counts.    ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
   Test results for c:\v.vol
 Chi-square with 5^5-5^4=2500 d.of f. for sample size:2560000
                              chisquare  equiv normal  p-value
  Results fo COUNT-THE-1's in successive bytes:
 byte stream for c:\v.vol         2440.35      -.844      .199470
 byte stream for c:\v.vol         2567.70       .957      .830831

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::     This is the COUNT-THE-1's TEST for specific bytes.        ::
     :: Consider the file under test as a stream of 32-bit integers.  ::
     :: From each integer, a specific byte is chosen , say the left-  ::
     :: most::  bits 1 to 8. Each byte can contain from 0 to 8 1's,   ::
     :: with probabilitie 1,8,28,56,70,56,28,8,1 over 256.  Now let   ::
     :: the specified bytes from successive integers provide a string ::
     :: of (overlapping) 5-letter words, each "letter" taking values  ::
     :: A,B,C,D,E. The letters are determined  by the number of 1's,  ::
     :: in that byte::  0,1,or 2 ---> A, 3 ---> B, 4 ---> C, 5 ---> D,::
```

```
     :: and  6,7 or 8 ---> E.  Thus we have a monkey at a typewriter  ::
     :: hitting five keys with with various probabilities::  37,56,70,::
     :: 56,37 over 256. There are 5^5 possible 5-letter words, and    ::
     :: from a string of 256,000 (overlapping) 5-letter words, counts ::
     :: are made on the frequencies for each word. The quadratic form ::
     :: in the weak inverse of the covariance matrix of the cell      ::
     :: counts provides a chisquare test::  Q5-Q4, the difference of  ::
     :: the naive Pearson  sums of (OBS-EXP)^2/EXP on counts for 5-   ::
     :: and 4-letter cell counts.                                     ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
 Chi-square with 5^5-5^4=2500 d.of f. for sample size: 256000
                      chisquare  equiv normal  p value
  Results for COUNT-THE-1's in specified bytes:
           bits  1 to  8  2479.63      -.288      .386642
           bits  2 to  9  2585.55      1.210      .886824
           bits  3 to 10  2446.78      -.753      .225815
           bits  4 to 11  2381.28     -1.679      .046585
           bits  5 to 12  2544.62       .631      .736009
           bits  6 to 13  2393.79     -1.502      .066549
           bits  7 to 14  2623.91      1.752      .960146
           bits  8 to 15  2654.46      2.184      .985534
           bits  9 to 16  2601.67      1.438      .924757
           bits 10 to 17  2476.00      -.339      .367142
           bits 11 to 18  2393.52     -1.506      .066058
           bits 12 to 19  2563.43       .897      .815152
           bits 13 to 20  2500.38       .005      .502154
           bits 14 to 21  2480.90      -.270      .393519
           bits 15 to 22  2576.16      1.077      .859279
           bits 16 to 23  2424.84     -1.063      .143923
           bits 17 to 24  2578.27      1.107      .865832
           bits 18 to 25  2497.15      -.040      .483910
           bits 19 to 26  2662.98      2.305      .989413
           bits 20 to 27  2533.69       .476      .683112
           bits 21 to 28  2449.91      -.708      .239374
           bits 22 to 29  2459.31      -.575      .282480
           bits 23 to 30  2498.34      -.023      .490652
           bits 24 to 31  2429.01     -1.004      .157710
           bits 25 to 32  2527.17       .384      .649592

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::                THIS IS A PARKING LOT TEST                    ::
     :: In a square of side 100, randomly "park" a car---a circle of  ::
     :: radius 1.   Then try to park a 2nd, a 3rd, and so on, each    ::
     :: time parking "by ear".  That is, if an attempt to park a car  ::
     :: causes a crash with one already parked, try again at a new    ::
     :: random location. (To avoid path problems, consider parking    ::
     :: helicopters rather than cars.)   Each attempt leads to either ::
     :: a crash or a success, the latter followed by an increment to  ::
     :: the list of cars already parked. If we plot n:  the number of ::
     :: attempts, versus k::  the number successfully parked, we get a::
     :: curve that should be similar to those provided by a perfect   ::
     :: random number generator.  Theory for the behavior of such a   ::
     :: random curve seems beyond reach, and as graphics displays are ::
     :: not available for this battery of tests, a simple characteriz ::
     :: ation of the random experiment is used: k, the number of cars ::
     :: successfully parked after n=12,000 attempts. Simulation shows ::
     :: that k should average 3523 with sigma 21.9 and is very close  ::
     :: to normally distributed.  Thus (k-3523)/21.9 should be a st-  ::
     :: andard normal variable, which, converted to a uniform varia-  ::
     :: ble, provides input to a KSTEST based on a sample of 10.      ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
          CDPARK: result of ten tests on file c:\v.vol
           Of 12,000 tries, the average no. of successes
                should be 3523 with sigma=21.9
           Successes: 3548    z-score:  1.142 p-value: .873180
           Successes: 3549    z-score:  1.187 p-value: .882429
           Successes: 3515    z-score:  -.365 p-value: .357445
           Successes: 3517    z-score:  -.274 p-value: .392053
           Successes: 3520    z-score:  -.137 p-value: .445521
           Successes: 3512    z-score:  -.502 p-value: .307734
           Successes: 3562    z-score:  1.781 p-value: .962529
           Successes: 3510    z-score:  -.594 p-value: .276387
           Successes: 3544    z-score:   .959 p-value: .831196
           Successes: 3528    z-score:   .228 p-value: .590298

          square size   avg. no.  parked   sample sigma
            100.           3530.500       17.668
           KSTEST for the above 10: p=  .609155

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::                THE MINIMUM DISTANCE TEST                     ::
     :: It does this 100 times::   choose n=8000 random points in a   ::
     :: square of side 10000.  Find d, the minimum distance between   ::
     :: the (n^2-n)/2 pairs of points.  If the points are truly inde- ::
     :: pendent uniform, then d^2, the square of the minimum distance ::
     :: should be (very close to) exponentially distributed with mean ::
     :: .995 .  Thus 1-exp(-d^2/.995) should be uniform on [0,1) and  ::
     :: a KSTEST on the resulting 100 values serves as a test of uni- ::
     :: formity for random points in the square. Test numbers=0 mod 5 ::
     :: are printed but the KSTEST is based on the full set of 100    ::
     :: random choices of 8000 points in the 10000x10000 square.      ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
              This is the MINIMUM DISTANCE test
             for random integers in the file c:\v.vol
     Sample no.    d^2     avg     equiv uni
           5     .5414    .2748     .419667
          10     .5014    .4398     .395868
          15     .0915    .5129     .087830
          20    1.6190    .8308     .803504
          25     .5839    .8505     .443909
          30    4.4954    .9124     .989089
          35     .0738    .8088     .071447
          40    1.7015    .8514     .819141
          45     .5949    .8813     .450012
          50    1.7453    .9618     .826930
          55     .0241    .9096     .023953
          60    1.4153    .9567     .758870
          65     .9082    .9279     .598598
          70     .8223    .9104     .562402
          75    1.5852    .9133     .796728
          80    1.4892    .8908     .776139
          85     .9590    .8869     .618579
          90    1.3030    .8966     .730059
          95     .5489    .8934     .424020
         100     .4777    .9033     .381279
     MINIMUM DISTANCE TEST for c:\v.vol
          Result of KS test on 20 transformed mindist^2's:
                                  p-value= .456306

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::                  THE 3DSPHERES TEST                            ::
     :: Choose  4000 random points in a cube of edge 1000.  At each   ::
     :: point, center a sphere large enough to reach the next closest ::
     :: point. Then the volume of the smallest such sphere is (very   ::
     :: close to) exponentially distributed with mean 120pi/3.  Thus  ::
     :: the radius cubed is exponential with mean 30. (The mean is    ::
     :: obtained by extensive simulation).  The 3DSPHERES test gener- ::
     :: ates 4000 such spheres 20 times.  Each min radius cubed leads ::
     :: to a uniform variable by means of 1-exp(-r^3/30.), then a     ::
     ::  KSTEST is done on the 20 p-values.                           ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
               The 3DSPHERES test for file c:\v.vol
 sample no:  1     r^3=  27.580     p-value= .60122
 sample no:  2     r^3=  14.087     p-value= .37473
 sample no:  3     r^3= 106.000     p-value= .97079
 sample no:  4     r^3=   7.416     p-value= .21901
 sample no:  5     r^3=   6.216     p-value= .18715
 sample no:  6     r^3=  66.791     p-value= .89208
 sample no:  7     r^3=  34.621     p-value= .68464
 sample no:  8     r^3=  25.765     p-value= .57635
 sample no:  9     r^3=   9.521     p-value= .27193
 sample no: 10     r^3=   2.949     p-value= .09361
 sample no: 11     r^3=  40.875     p-value= .74398
 sample no: 12     r^3=   4.795     p-value= .14772
 sample no: 13     r^3=  20.089     p-value= .48810
 sample no: 14     r^3=  53.067     p-value= .82948
 sample no: 15     r^3=  32.826     p-value= .66520
 sample no: 16     r^3=  35.976     p-value= .69857
 sample no: 17     r^3=  22.054     p-value= .52056
 sample no: 18     r^3=   2.463     p-value= .07881
 sample no: 19     r^3=  65.466     p-value= .88721
 sample no: 20     r^3=  17.486     p-value= .44171
  A KS test is applied to those 20 p-values.
-------------------------------------------------------
       3DSPHERES test for file c:\v.vol            p-value= .011014
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::      This is the SQEEZE test                                  ::
     ::  Random integers are floated to get uniforms on [0,1). Start- ::
     ::  ing with k=2^31=2147483647, the test finds j, the number of  ::
     ::  iterations necessary to reduce k to 1, using the reduction   ::
     ::  k=ceiling(k*U), with U provided by floating integers from    ::
     ::  the file being tested.  Such j's are found 100,000 times,    ::
     ::  then counts for the number of times j was <=6,7,...,47,>=48  ::
     ::  are used to provide a chi-square test for cell frequencies.  ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
           RESULTS OF SQUEEZE TEST FOR c:\v.vol
         Table of standardized frequency counts
     ( (obs-exp)/sqrt(exp) )^2
        for j taking values <=6,7,8,...,47,>=48:
     -.1     -.3     2.0    -1.1      .8     1.4
      .7      .6    -1.5      .9     -.1      .4
    -1.9    -1.8      .9     1.8      .1    -1.1
     -.1     -.1      .4     1.3     1.0    -1.2
    -1.5     1.0     -.2      .4      .8     -.1
      .2    -1.4      .6    -1.0    -1.0     -.8
    -1.4    -1.0      .9      .4      .1     1.0
     -.1
           Chi-square with 42 degrees of freedom: 42.510
              z-score=   .056  p-value= .550980
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
     ::            The  OVERLAPPING SUMS test                        ::
     :: Integers are floated to get a sequence U(1),U(2),... of uni-  ::
     :: form [0,1) variables.  Then overlapping sums,                 ::
     ::   S(1)=U(1)+...+U(100), S2=U(2)+...+U(101),... are formed.    ::
     :: The S's are virtually normal with a certain covariance mat-   ::
     :: rix.  A linear transformation of the S's converts them to a   ::
     :: sequence of independent standard normals, which are converted ::
     :: to uniform variables for a KSTEST. The  p-values from ten     ::
     :: KSTESTs are given still another KSTEST.                       ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
               Test no.  1      p-value  .639054
               Test no.  2      p-value  .192778
               Test no.  3      p-value  .254935
               Test no.  4      p-value  .745106
               Test no.  5      p-value  .715922
               Test no.  6      p-value  .724297
               Test no.  7      p-value  .357862
               Test no.  8      p-value  .581758
               Test no.  9      p-value  .502808
               Test no. 10      p-value  .874405
   Results of the OSUM test for c:\v.vol
        KSTEST on the above 10 p-values:  .459334

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     ::     This is the RUNS test.  It counts runs up, and runs down, ::
     :: in a sequence of uniform [0,1) variables, obtained by float-  ::
     :: ing the 32-bit integers in the specified file. This example   ::
     :: shows how runs are counted:  .123,.357,.789,.425,.224,.416,.95::
     :: contains an up-run of length 3, a down-run of length 2 and an ::
     :: up-run of (at least) 2, depending on the next values.  The    ::
     :: covariance matrices for the runs-up and runs-down are well    ::
     :: known, leading to chisquare tests for quadratic forms in the  ::
     :: weak inverses of the covariance matrices.  Runs are counted   ::
     :: for sequences of length 10,000.  This is done ten times. Then ::
     :: repeated.                                                     ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
          The RUNS test for file c:\v.vol
     Up and down runs in a sample of 10000
_____________________________________________
                 Run test for c:\v.vol       :
       runs up; ks test for 10 p's: .390983
     runs down; ks test for 10 p's: .905959
                 Run test for c:\v.vol       :
       runs up; ks test for 10 p's: .297555
     runs down; ks test for 10 p's: .417169

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
     :: This is the CRAPS TEST. It plays 200,000 games of craps, finds::
     :: the number of wins and the number of throws necessary to end  ::
     :: each game.  The number of wins should be (very close to) a    ::
     :: normal with mean 200000p and variance 200000p(1-p), with      ::
     :: p=244/495.  Throws necessary to complete the game can vary    ::
     :: from 1 to infinity, but counts for all>21 are lumped with 21. ::
     :: A chi-square test is made on the no.-of-throws cell counts.   ::
     :: Each 32-bit integer from the test file provides the value for ::
     :: the throw of a die, by floating to [0,1), multiplying by 6    ::
     :: and taking 1 plus the integer part of the result.             ::
     ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
                 Results of craps test for c:\v.vol
    No. of wins:  Observed Expected
                               98345    98585.86
                  98345= No. of wins, z-score=-1.077 pvalue= .14068
     Analysis of Throws-per-Game:
   Chisq=  16.08 for 20 degrees of freedom, p=  .28866
                 Throws Observed Expected  Chisq     Sum
                    1    66641    66666.7    .010     .010
                    2    37438    37654.3   1.243    1.253
                    3    26875    26954.7    .236    1.489
                    4    19613    19313.5   4.646    6.134
                    5    13817    13851.4    .086    6.220
                    6     9964     9943.5    .042    6.262
                    7     7121     7145.0    .081    6.343
                    8     5262     5139.1   2.941    9.283
                    9     3703     3699.9    .003    9.286
                   10     2661     2666.3    .011    9.296
                   11     1943     1923.3    .201    9.497
                   12     1361     1388.7    .554   10.052
                   13      960     1003.7   1.904   11.955
                   14      720      726.1    .052   12.007
                   15      535      525.8    .160   12.167
                   16      374      381.2    .134   12.301
                   17      281      276.5    .072   12.373
                   18      177      200.8   2.828   15.201
                   19      157      146.0    .831   16.032
                   20      106      106.2    .000   16.032
                   21      291      287.1    .053   16.085
              SUMMARY  FOR c:\v.vol
                  p-value for no. of wins: .140681
                  p-value for throws/game: .288661

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

```
Results of DIEHARD battery of tests sent to file out.txt
```

For more information: http://www.pmc-ciphers.com

Microsoft, the Office logo, Outlook, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.
Company and product names mentioned herein may be the trademarks of their respective owners.

-o-

Installation of the TurboCrypt control panel

Launch "setup.exe" from the root directory of the TurboCrypt CD-ROM or from the extracted archive and follow the instructions on the screen. When TurboCrypt runs for the first time, it will ask to automatically install the encryption driver. If an old driver is present in the system, the old driver will be replaced by the new one automatically. Without the encryption driver the software cannot operate.

Automatic driver installation

The TurboCrypt volume encryption tool is based on a software driver which emulates removable disk drives. Your operating system must be Windows 2000, XP or Windows 2003 Server or later!

The TurboCrypt encryption driver is a state-of-the-art plug-and-play NT5.x driver. TurboCrypt can install this driver automatically! During automatic installation, a dialog pops up with which you can choose a compatible driver. The current version is 2118. With the "Have disk.." button, you can specify a file path where the new driver volcrypt.inf file is located. This button may only be needed if you want to install an updated driver.

Manual driver installation

It is very unlikely that you will ever need to do this manually as the automatic driver install/update functions work well in over 20.000 installations.
To install the driver manually, please follow these steps (shown for Windows 2000; Windows XP additionally checks if the hardware is already installed: As there is no new hardware, choose "New hardware is already installed"):

1. Open the control panel
2. Choose "Add New Hardware"
3. Click Next two times, select "No, I want to select hardware from a list" and click Next
4. Select "Other devices" and click Next
5. Click "Have Disk..." and browse to the help directory of the TurboCrypt CD-ROM, select "volcrypt.inf" and click "OK"
6. One model is listed from the found installation INF file. Select it and click Next. Select "Finish" to complete installation.

There is no need to reboot the system. The driver starts immediately after installation.

Deinstalling TurboCrypt

The encryption driver and the control panel software deinstall separately. The control panel is easily deinstalled by choosing add/remove software in the Windows Control Panel.

The encryption driver can either be left in the system or it can be deinstalled as described below:
1. Open the control panel
2. Choose "System"
3. Select the "Hardware" tab sheet
4. Click at the "Device Manager" button - a window showing the list of devices opens
5. Double-click at "Volume Encryption Devices" and select the "Ultra-secure 512 bit volume encryption driver"

6. Hit the delete button and confirm with "yes"

There is no need to reboot the system. The driver terminates automatically and, if there are still encrypted volumes open, a temporary instance closes them at system shutdown.

-o-

The Control Panel

The TurboCrypt Control Panel is arranged in three areas:
On the left side there is volume encryption, trace deletion, unused disk space wipe, file shredding, e-mail encryption and a check for program updates accessible. In the center on top there are encrypted volumes and encrypted partitions displayed. Below there are all recognized physical storage devices listed, which could contain additional encrypted volumes. The right side contains system functionality to control volume encryption: Add encrypted volume, Import encrypted volume, Mount volume, Lock volume, Change volume name, Change password, Remove volume, Options, Image files for backup.

-o-

Menu on the right side

The following options are available by clicking on the left mouse button:

- Add encrypted volume
- Import encrypted volume
- Mount volume
- Lock volume
- Change volume name

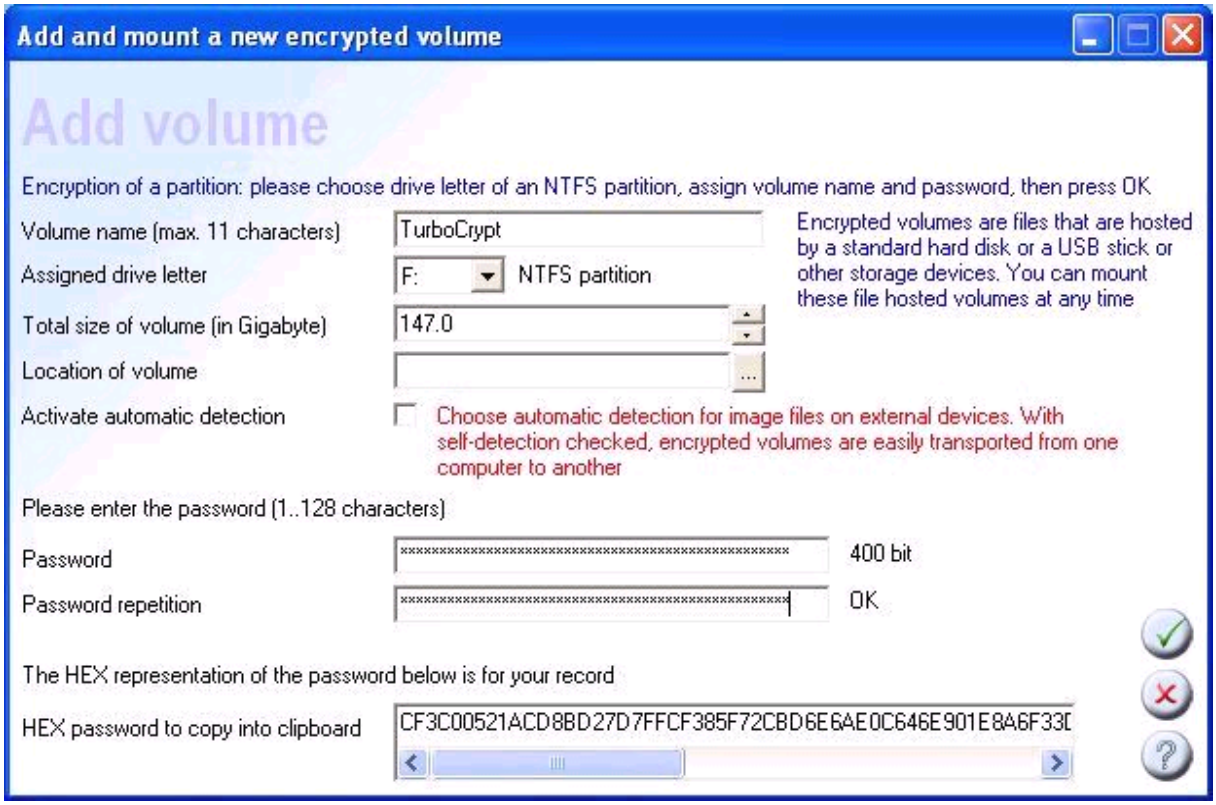
- Change password
- Remove volume

- Options
- Image files for backup

-o-

Add encrypted volume

When clicking at the button "Add encrypted volume", the following dialog box appears:



Volume name (choose freely)

Enter a drive label of your choice. This name will later appear in the list of drives (for non-permanent drives only).

Assigned drive letter

Please choose a drive letter from the pull-down menu. The pull-down menue contains only currently free drive letters. The Enterprise Edition additionally displays NTFS partitions which can be encrypted. Although it is possible to encrypt NTFS partitions on which programs are installed, it is not recommended to do so. If unmounted, the system might try to load DLLs dynamically and this will of course fail. So better use this feature with partitions which only contain data rather than software installations.

Activate automatic detection

If you set this checkbox to the checked state, the volume which is to be created will be identified by the built-in automatic drive scanner and is added to the list of available volumes. A drive letter is chosen dynamically each time the volume is mounted.
This feature is mainly intented for the creation of volumes on removable drives like external hard disks or USB memory sticks. Automatic detection is always activated for volume files which are created on removable drives.
This feature cannot be activated for raw NTFS partitions (applies only to the Enterprise Edition)

Location of volume

Please enter the file path where you want to physically store the new volume file.
In order to choose a location for the creation of the encrypted volume image file, click at the

button on the right side of the edit box.
A directory picker dialog opens and you can search a suitable location.
This feature is deactivated for the encryption of raw NTFS partitions (applies only to the Enterprise Edition).

Total size of volume
The volume size can be chosen manually or by using the small up-and down arrows which are on the right side of the volume size edit box.
Raw partitions have a fixed size. Consequently, this feature is deactivated for the encryption of raw NTFS partitions (applies only to the Enterprise Edition).

Password
Here you can enter the password to protect your encrypted volume. Please choose a long password like "Wa4X+g2#csdf89#2bDWXvtzks92m#fk6y10h". With the length of the password and the degree of uniqueness you indirectly choose the quality of the encryption: A short password like "Wa", corresponds with about 12 .. 16 bit encryption strength. Such simple passwords are very easy to crack!
TurboCrypt maps all password entries to 512 bit long binary representations. Each character of your entry adds 6 ... 8 password bits. Consequently, after entering approximately 80 characters, no more effective password information is added. An exception to this rule are 128 characters long HEX passwords (must be set accordingly in the Options dialog). TurboCrypt expects these passwords to consist solely of hexadecimal characters. Each character contains exactly 4 password bits. Thus, after entering all 128 characters, 512 password bits are specified without executing the password mapping (hashing) process. This is a useful feature for users who utilize smart cards for password entry and who keep printed password information in two different places (common procedure in banks).

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

Password repetition
The chosen password must be entered in this edit box once more. The software checks boths entries for being identical and thus prevents typing errors.

HEX password:
This edit box displays the internal 512 bit representation of your password. It can be copied to the clipboard. Banks use this feature to give two different persons half of the password. Each of them cannot access the data, but together they can.

On the right side of this dialog there are three buttons. Here's a description of their functionality:

OK

After specifying all required parameters, click at the OK button to save the settings. The OK button remains disabled as long as not all required

Cancel

If you have entered the wrong data or if you want to quit this dialog, hit the Cancel button.

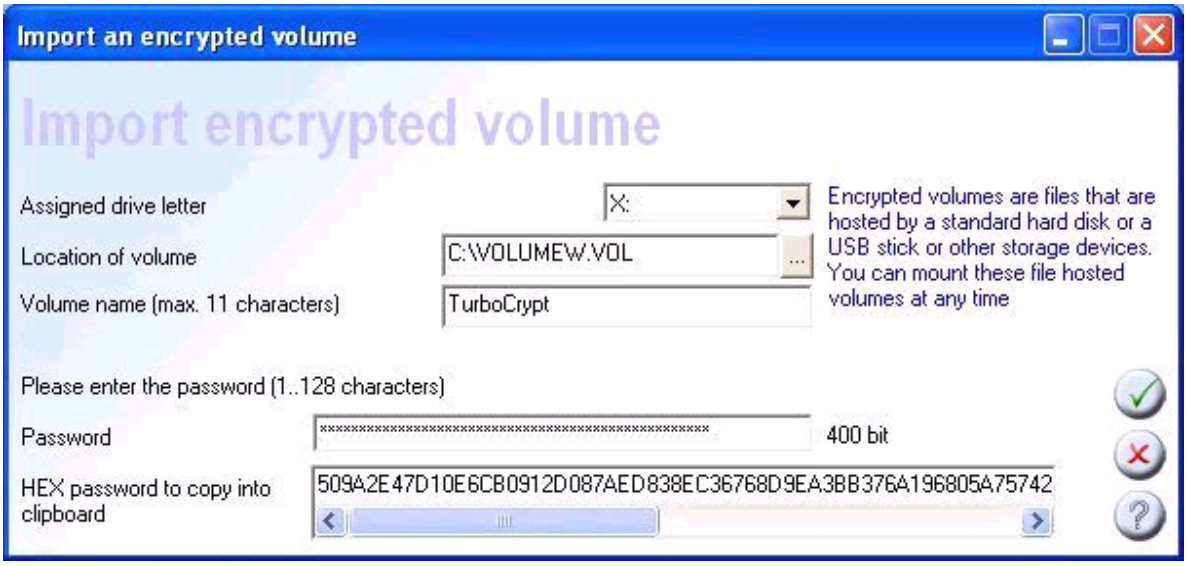
Help

By clicking at this button, a dialog containing support data opens and you get more information.

-o-

Import encrypted volume

In order to import a volume image file of an existing encrypted drive in your system, click at Import volume. The following dialog pops up:



Assigned drive letter
With the help of a pull-down menu you can choose from a list of free drive letters one which you want to assign the volume to. Usually the first available drive letter is already displayed.

Location of volume
Please specify the file path of the volume file which you want to mount to the file system here. In order to choose an encrypted volume image file, click at the button on the right side of the edit box.
A directory picker dialog opens and you can search for volume files (default file extension is .vol).

Note: It is possible to import volume files that are stored on CD-ROMs or DVD-ROMs. The encrypted volume will be attached as read-only file volume only, although.
If the CD-ROM is not in place when TurboCrypt control panel is launched, the volume definition will be deleted automatically from the Windows Registry.

Users who are familiar with regedit.exe find all volume definitions under

HKEY_CURRENT_USER\Software\DiskEncryption\Drives.
WARNING: Changing information stored in the Registry can cause severe malfunction of a computer!

Volume name (choose freely)
Enter a drive label of your choice. This name will later appear in the list of drives.

Password
Please enter here the password with which you have already encrypted the volume.

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

OK

After entering all the data to import the encrypted volume, simply click at the OK button to save the settings and to create the encrypted volume. After the creation process is finished, you can use the new drive.

Cancel

If you have entered the wrong data or if you want to go back to the previous dialog, hit the Cancel button. The new settings will be discarded.

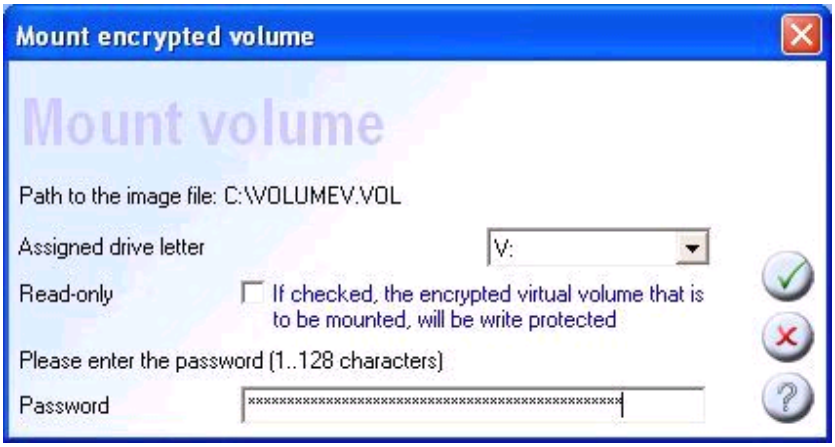
Help

By clicking at this button, a dialog containing support data opens and you get more information.

-o-

Mount volume

Unmounted volumes are crossed out in the list of drives in the main window. If you want to mount a currently locked volume by clicking at the volume with the left mouse button and then clicking at the "Mount volume" button, the following dialog pops up:



Assigned drive letter
Choose from the pull-down menu on the right side a free drive letter.

Read-only
Set this box to the checked state if you want to prevent overwriting or deletion of data on your encrypted volume once it is mounted. The volume can then be used just like a CD-ROM.

Password
Please enter here the password with which you have already encrypted the volume.

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

OK

After entering all the data to mount an encrypted volume, simply click at the OK button to save the settings and to perform the desired function. After the mounting process is finished, you can use the drive.

Cancel

If you have entered the wrong data or if you want to go back to the previous dialog, hit the Cancel button. The new settings will be discarded.

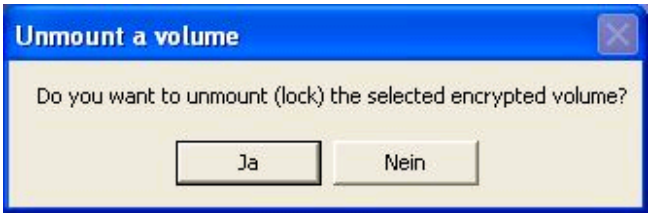
Help

By clicking at this button, a dialog containing support data opens and you get more information.

-o-

Unmount (lock) volumes

In order to lock a mounted encrypted volume, mark the drive in the listbox of the TurboCrypt control panel (the main window) and click at "Lock volume". The following confirmation dialog pops up:



Before you confirm with "Yes" (or „Ja“ if you happen to have a German version of the Operating System), please close all open files on this volume. If there's still a file open or if Windows Explorer points at the drive which is to be unmounted, the software will not be able to unmount the volume. If you are not sure, hit "no".

All mounted encrypted volumes will be closed automatically at system shutdown.

Please note that once you have confirmed that you want to continue the operation by clicking at "Yes", only after entering the correct password in the "Mount" dialog you will be able to access data which is stored on the encrypted volume.

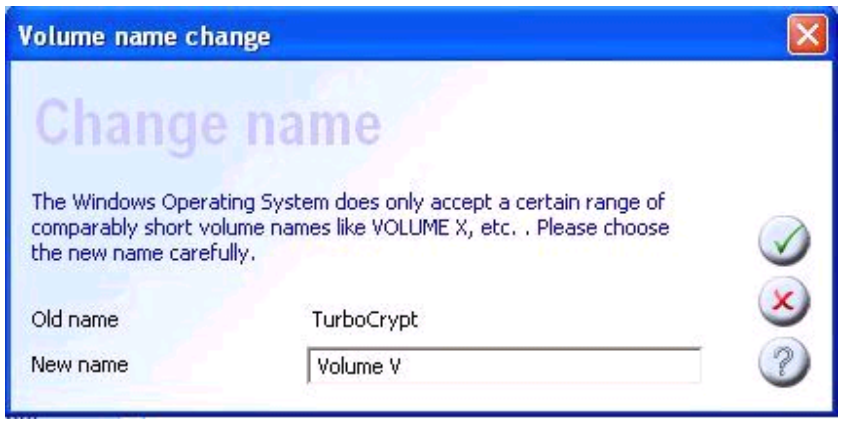
! If you forget your password, you will never be able to access your data again.

-o-

Change volume name

The volume name assigned by the user can be changed any time for mounted encrypted volumes, as well as for NTFS partitions. Mark a mounted volume or NTFS partition in the listbox of the TurboCrypt control panel (the main window) and click at "Change name".

The following dialog pops up:



Old name
The old name that had been assigned by the user is displayed here.

New name
Volume names can be up to 11 characters long. The Operating System is a bit picky. After pressing the OK button, the new name is assigned to the encrypted volume. If this operation fails, the dialog box will pop up again and you'll can try other names.

OK

After entering all the data to change the user-defined name, click at the OK button to save the settings and to perform the desired function. If the file system is unable to assign the new name to the mounted volume, you will be asked to correct your settings. After the process is finished, you can e.g. see the new name in Windows Explorer.

Cancel

If you have entered wrong data or if you want to go back to the previous dialog, hit the Cancel button. The new settings will be discarded.

Help

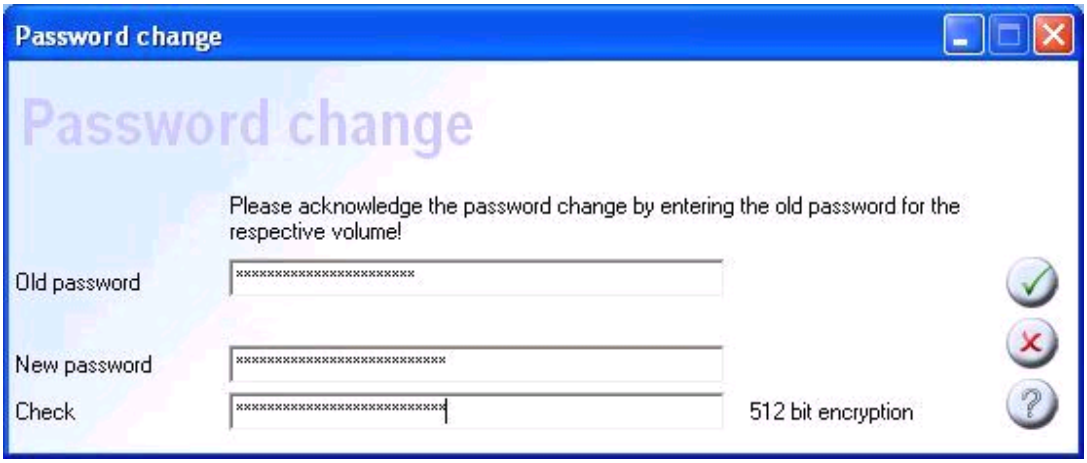
By clicking at this button, a dialog containing support data opens and you get more information.

-o-

Change password

In order to protect an encrypted volume with a new password, mark an unmounted drive in the listbox of the TurboCrypt control panel (the main window) and click at "Change password". If you want to change the password of a mounted drive, please lock (unmount) it first, then click at "Change password". Password changes are executed very rapidly because a so-called disk key is only re-encrypted. Please note that this doesn't apply to raw NTFS partitions. Unlike file-hosted volumes, these encrypted storage devices are directly encrypted with the password hash. Changing the password consequently implies a complete re-encryption of the partition!

The following dialog pops up:



Old password
Please enter here the password with which the volume is momentarily encrypted.

New password
Here the new password must be entered. It is highly advisable to choose a long password like "Wa4X+g2#csdf89#2bDWXvtzks92m#fk6y10h". With the length of the password and the degree of uniqueness you indirectly choose the quality of the encryption: A short password like "Wa", corresponds to about 12 .. 16 bit encryption! Such simple passwords are very easy to crack.

TurboCrypt maps all password entries to 512 bit long binary representations. Each character of your entry adds 6 ... 8 password bits. Consequently, after entering approximately 80 characters, no more effective password information is added. An exception to this rule are 128 characters long HEX passwords (must be set accordingly in the Options dialog). TurboCrypt expects these passwords to consist solely of hexadecimal characters. Each character contains exactly 4 password bits. Thus, after entering all 128 characters, 512 password bits are specified without executing the password mapping (hashing) process.

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

Check (password repetition)
The chosen password must be entered in this edit box once more. The software checks boths entries for being identical and thus prevents typing errors.

OK

After entering all the data to change the password, click at the OK button to save the settings and to perform the desired function. If the two lines containing the new password are not identical, you will be asked to correct your settings. After the process has finished, you can only mount the drive with the changed (new) password.

Cancel

If you have entered wrong data or if you want to go back to the previous dialog, hit the Cancel button. The new settings will be discarded.

Help

By clicking at this button, a dialog containing support data opens and you get more information.

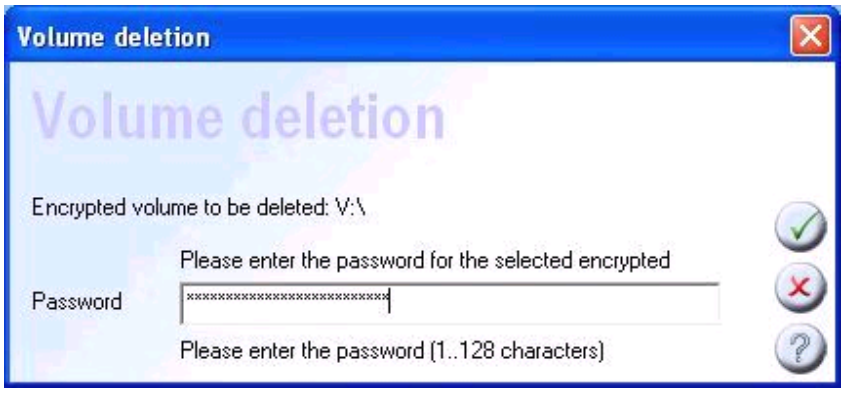
-o-

Remove volume

With this function you can delete and secure wipe an encrypted volume

! Removing an encrypted volume destroys secured data on the selected volume.

Mark an unmounted drive in the listbox of the TurboCrypt control panel (the main window) and click at "Remove Volume". The following dialog pops up:



Password
Please enter here the password with which you have already encrypted the volume.

OK

After entering all relevant data to remove the selected volume, click at the OK button to save the settings and to perform the desired function. If you have activated the secure wipe function in the Options menu, the volume will be overwritten with "noise" prior to deletion.

Cancel

If you have entered wrong data or if you want to go back to the previous dialog, hit the Cancel button. The new settings will be discarded.

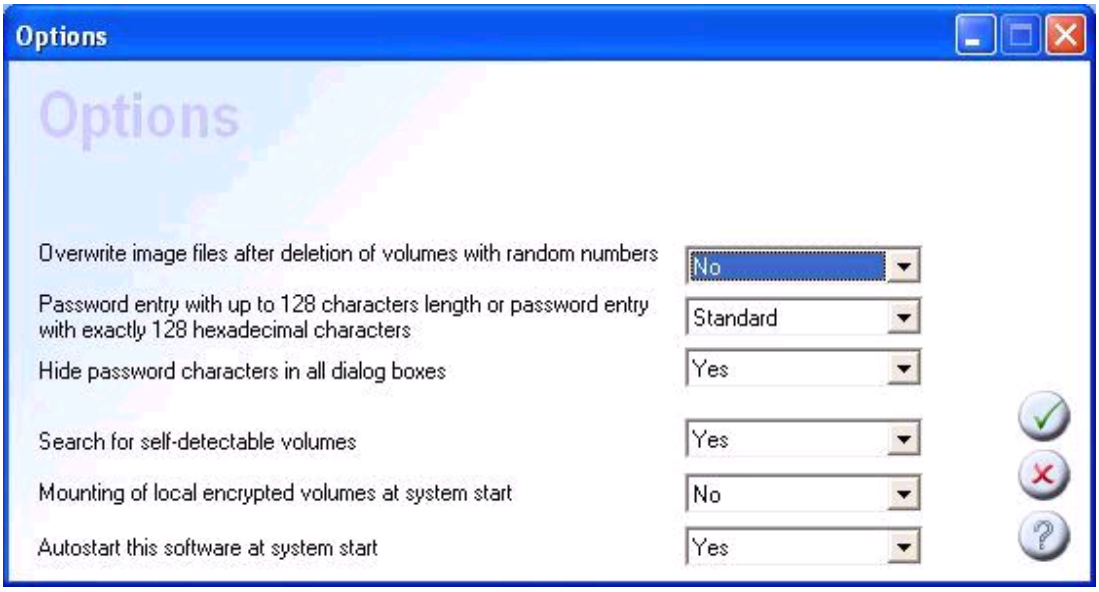
Help

By clicking at this button, a dialog containing support data opens and you get more information.

-o-

Options

The following dialog opens when you click at the "Options" button in the main window:



With this dialog you can set options which affect the appearance and characteristics of the program. You can increase the security when deleting a volume image file, choose which type of password entry you want to use and you can customize system settings.

Changing standard options is only advisable for advanced users. Here's a short description of the individual topics:

Overwrite image files after deletion of volumes with random numbers

In case you want to remove encrypted volumes and you want to make forensic analysis of your computer impossible, choose "Yes". If your password might be known by other people, it is highly advisable to choose "Yes".

Password entry with up to 128 characters length or password entry with exactly 128 hexadecimal characters

TurboCrypt supports standard password entry with password lengths ranging from 1 character up to 128 characters. These entries are mapped internally by a polymorphic hashing algorithm to exactly 512 bit.

Alternatively, passwords can be entered as 128 characters long hexadecimal numbers. By doing this, the hash algorithm is bypassed.

When new encrypted drives are created, the 128 character hex number is displayed. As an example, it could be copied into a text document, split into two pieces and subsequently stored in two different files on two different computers. If the password is lost somehow, the two pieces of the hex representation could be concatenated and pasted into the password edit line in the "Mount" dialog in order to be able to access the encrypted data which would otherwise be locked permanently.

Hide password characters in all dialog boxes

If set to "Yes", each password characters is displayed as an asterisk . This makes visual spying on your passwords more difficult.

Search for self-detectable volumes
If set to "Yes", encrypted volumes residing on removable drives are detected automatically and are displayed in the main window.

Mounting of local encrypted volumes at system start
On system start the TurboCrypt control panel is launched automatically and the user is asked for the passwords of the encrypted volumes and raw devices which are known to the system. If no password is entered or if it is incorrect, the affected encrypted volume is not mounted to the file system..

Autostart this software at system start
TurboCrypt is launched at system start if this option is active.

OK

After adjusting the settings, click at the OK button to save the settings.

Cancel

If you have entered the wrong data or if you want to go back to the previous dialog, hit the Cancel button. The new settings will be discarded.

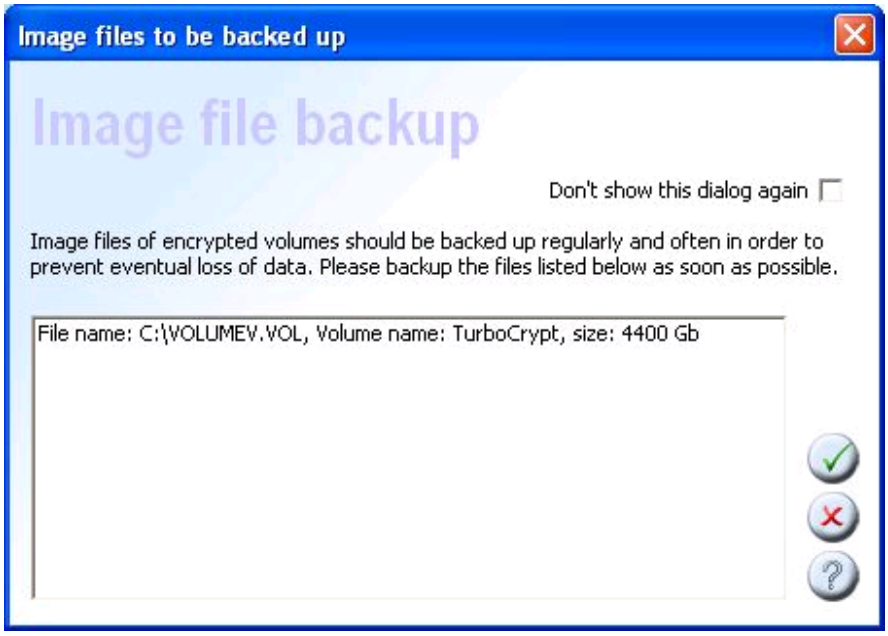
Help

By clicking at this button, a dialog containing support data opens and you get more information.

-o-

Image files for backup

The following dialog opens when you click at the "Image files for backup" button in the main window:



This dialog displays file names and file sizes of all currently known encrypted volumes. File hosted volumes are very convenient for backup (must altough be done "manually"): They can be stored on DVD-ROM's, hard disks or USB sticks and they can even be imported from these media in case you need to access your backups later.
The image file of a 4.3GB large encrypted volume fits completely on a DVD-ROM. It is the preferred volume size if you prefer to make backups on DVD's. For hard disk backups, any practical volume size is suitable.
Don't show this dialog again
this dialog pops up from time to time to remind users to make backups of their image file(s). In case you find this dialog annoying, click at this checkbox and then click ot the OK button.

OK

After adjusting the settings, click at the OK button to save the settings.

Cancel

If you have entered the wrong data or if you want to go back to the previous dialog, hit the Cancel button. The new settings will be discarded.

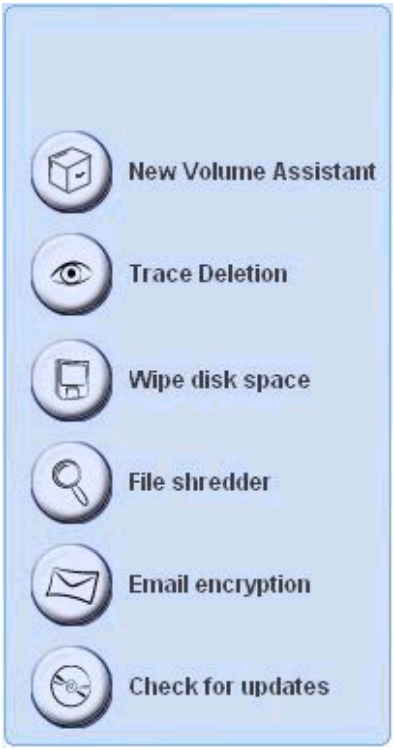
Help

By clicking at this button, a dialog containing support data opens and you get more information.

-o-

Menu on the left side

The following options are available by clicking on the left mouse button:



- New Volume Assistant
- Trace Deletion
- Wipe (unused) disk space
- File shredder
- Email encryption
- Check for program updates

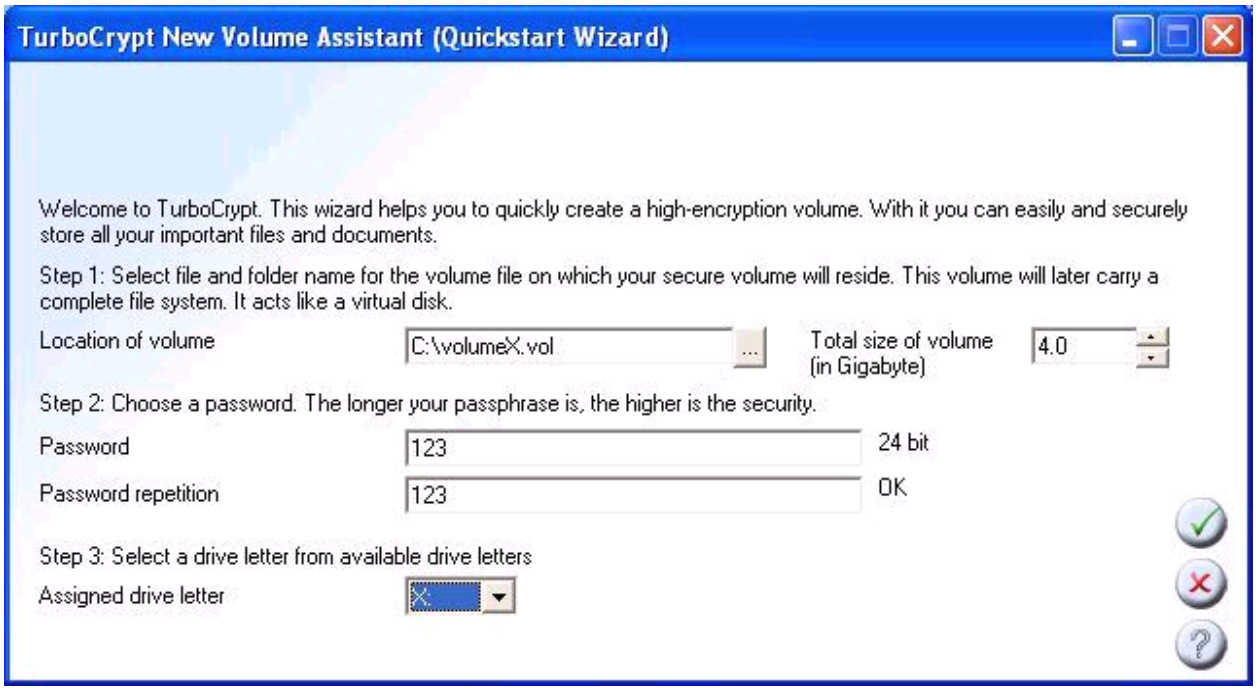
-o-

New Volume Assistant

The New Volume Assistant (quickstart wizard) is intended to ease first time use of TurboCrypt. The wizard automatically pops up if there are not yet encrypted volumes present on your system, but it is also available from the menu on the left side of the main window or from menu that pops on left mouse button clich at the vault sysmbol in the main window.

The New Volume Assistant helps users to add a new encrypted volume on a local drive or a transportable drive, e.g. a USB stick.

The New Volume Assistant helps users to add a new encrypted volume on a local drive or a transportable drive, e.g. a USB stick.



The wizard performs three steps:

1.) Location, name and size of image file which is later to be mounted to the file system and which hosts data that is stored to the encrypted volume

2.) Password entry: A password of your choice is to be entered in both edit box (the second entry is needed to check for misspelling). Long passwords like "Wa4X+g2#csdf89#2bDWXvtzks92m#fk6y10h" are generally more suitable to protect your data than short and simply ones like "abcdef".

With the length of the password and the degree of uniqueness you indirectly choose the quality of the encryption: A short password like "Wa", corresponds to about 12 .. 16 bit

encryption! Such simple passwords are very easy to crack.

TurboCrypt maps all password entries to 512 bit long binary representations. Each character of your entry adds 6 ... 8 password bits. Consequently, after entering approximately 80 characters, no more effective password information is added.

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

3.) Choice of drive character that is to be assigned to the new encrypted volume:
TurboCrypt will format and mount the newly created encrypted volume when the OK button is hit. The new volume will get the drive letter that you've chosen here. The wizard finally opens the new volume in Windows Explorer. All data that is stored on this volume is encrypted on the fly.

OK

After specifying all required parameters, click at the OK button to save the settings. The OK button remains disabled as long as not all required

Cancel

If you have entered the wrong data or if you want to quit this dialog, hit the Cancel button.

Help

By clicking at this button, a dialog containing support data opens and you get more information.

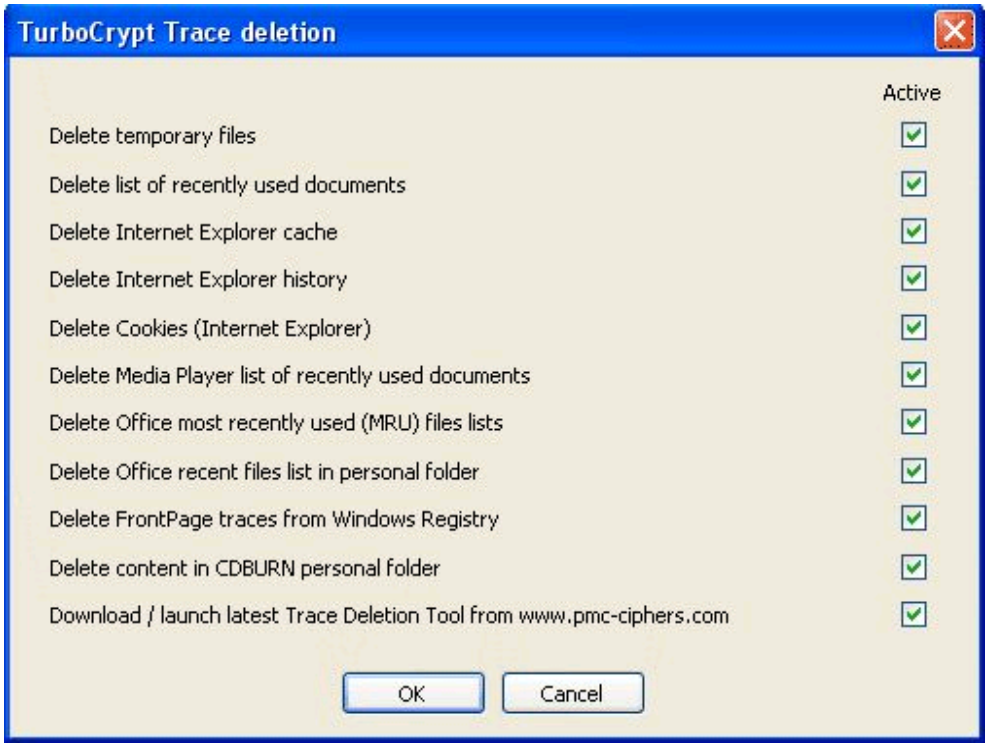
-o-

Trace Deletion

With this function, a number of items containing data about your habits and your work are deleted:

- Internet browser remains like cookies, history lists, etc.
- Temporary files
- List of recently used documents, Media Player- and Office MRU lists, Office recent files in personal folder
- FrontPage registry traces
- CDBURN personal folder

Additionally a function for automatic download and launch of the most up-to-date trace deletion extension utility is provided. Currently this tool cleans traces left by use of the RealPlayer.



-o-

Wipe Disk Space

When launching this function, a window pops up in which the user can choose on which writable medium he wants to wipe unused hard disk space. This function writes random numbers to the remaining storage capacity of the selected drive.
As the free capacity of modern hard disks is usually in the range of several 80 to 500 gigabytes, the implemented one-pass algorithm is optimized for speed.

-o-

File Shredder

A file section window pops up to select a file or folder to secure wipe (shred). The selected file(s) and/or folder(s) are overwritten using three different algorithms and they are subsequently deleted:

Wipe: The selected items are overwritten with real random numbers and after finishing this process, they are deleted.

This method is not approved by the DoD for sanitizing media that contain top secret information. This method does not take into account, that the hard disk head does not always fly over the center of the track.

Wipe DoD 5220.22-M: The selected items are overwritten three times prior to deleting them according to the standard 5220.22-M of the U.S. Department of Defense:
Step 1: Overwrite all addressable locations with a character
Step 2: Overwrite all addressable locations with with the complement of the previously written chracter
Step 3: Overwrite with a random character
Step 4: Verify the data which was previously written to the writable medium.

It should be noted that this method is not approved by the DoD for sanitizing media that contain top secret information

Wipe Gutmann: The selected items are overwritten 35 times prior to deleting them according to Peter Gutmann's method which was proposed in 1996 in his paper "Secure Deletion of Data from Magnetic and Solid-State Memory":

	Overwrite Data
Pass No.	**Data Written (Binary/Hexadecimal)**
1	Random numbers
2	Random numbers
3	Random numbers
4	Random numbers
5	01010101 01010101 01010101 0x55
6	10101010 10101010 10101010 0xAA
7	10010010 01001001 00100100 0x92 0x49 0x24
8	01001001 00100100 10010010 0x49 0x24 0x92
9	00100100 10010010 01001001 0x24 0x92 0x49
10	00000000 00000000 00000000 0x00
11	00010001 00010001 00010001 0x11
12	00100010 00100010 00100010 0x22
13	00110011 00110011 00110011 0x33
14	01000100 01000100 01000100 0x44
15	01010101 01010101 01010101 0x55
16	01100110 01100110 01100110 0x66
17	01110111 01110111 01110111 0x77
18	10001000 10001000 10001000 0x88
19	10011001 10011001 10011001 0x99
20	10101010 10101010 10101010 0xAA
21	10111011 10111011 10111011 0xBB
22	11001100 11001100 11001100 0xCC
23	11011101 11011101 11011101 0xDD
24	11101110 11101110 11101110 0xEE
25	11111111 11111111 11111111 0xFF

26	10010010 01001001 00100100	0x92 0x49 0x24
27	01001001 00100100 10010010	0x49 0x24 0x92
28	00100100 10010010 01001001	0x24 0x92 0x49
29	01101101 10110110 11011011	0x6D 0xB6 0xDB
30	10110110 11011011 01101101	0xB6 0xDB 0x6D
31	11011011 01101101 10110110	0xDB 0x6D 0xB6
32	Random numbers	
33	Random numbers	
34	Random numbers	
35	Random numbers	

The implemented algorithm is regarded as very secure throughout the industry. Although, when using this method, the consumption of a lot of processing time must be taken into account.

-o-

eMail encryption

Step 1:

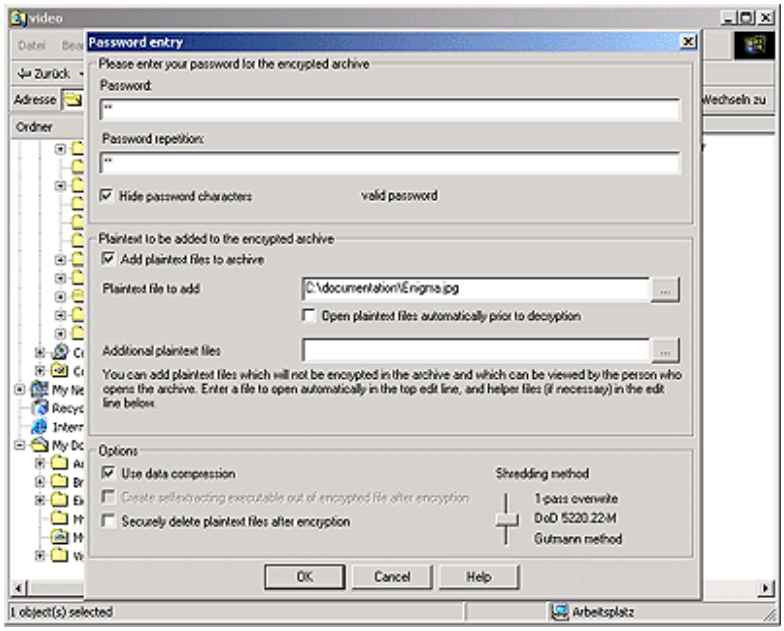
A file selection dialog pops up in which file(s)/folders(s) to be encrypted are selected.

Step 2:

Another file dialog appears in which file name and file path for the encrypted archive that is to be created and emailed is specified by the user.

Step 3:

The following password dialog pops up:



Password

Here you can enter the password to protect your encrypted volume. Please choose a long password like "Wa4X+g2#csdf89#2bDWXvtzks92m#fk6y10h".

With the length of the password and the degree of uniqueness you indirectly choose the quality of the encryption: A short password like "Wa", corresponds with about 12 .. 16 bit encryption strength. Such simple passwords are very easy to crack!
The TurboCrypt Shell Extension maps all password entries to 256 bit long binary representations. Each character of your entry adds 6 ... 8 password bits. Consequently, after entering approximately 40 characters, no more effective password information is added.

If users seek protection against automatic cipher breaking software, they should ONLY rely on very long passwords to take full advantage of the ultra-strong 256 bit encryption used to encrypt file archives. Today, 128 bit keys are regarded as totally safe for the next 100 years. With every additional key bit, attack security increases by two (=> 129 bit keys are safe for at least 200 years, etc.).

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again!

Hide password characters

In order to hide the password characters that are typed in, an asterik is displayed instead of the actual characters. This option can be switched on and off by clicking at the checkbox .

Plaintext files
When encrypting commercial data, it might be useful to add a plaintext file which contains information on the type of data that is stored in the archive. This plaintext file remains plaintext in the archive and can be viewed by anyone. Files containing sound or video can also be added to the encrypted archive, thus e.g. providing users with a preview. HTML pages sometimes require additional files to be stored with the main html file. These files are added by clicking at the button next to the second line ("Additional plaintext files"). All the files which are added here should originate from the same directory as the main plaintext file ("Plaintext file to add"). When a user later wants to view the plaintext files, all available files from the archive are stored in the same temporary folder!

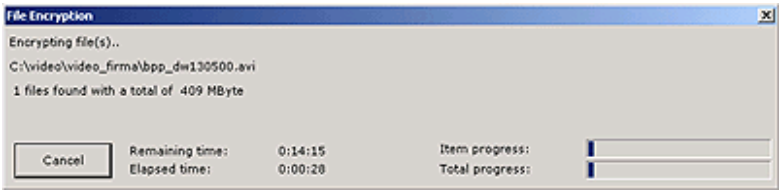
Options

Use data compression
If the checkbox "Use data compression" is checked, the encryption engine will try to compress the plaintext prior to the actual encryption process. If the compression succeeds, the compressed representation of the data is encrypted and saved to the archive. Although highly optimized compression algorithms are implemented in the software, it should be noted that data compression consumes most of the CPU time.

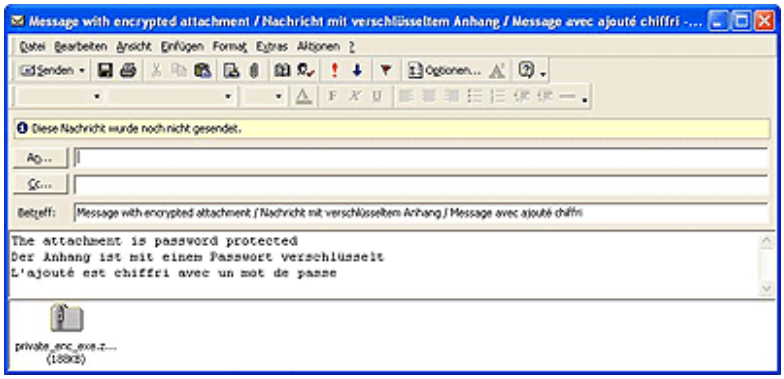
Create selfextracting executable out of encrypted file after encryption (needs NOT to be checked as this function is always used for email attachment encryption)
When setting this checkbox to the checked state, a program containing the decryption functions and the ciphertext file is additionally generated. The decryption functions add approximately 350kB to the file size. This function, which is only available for users who have purchased the software, enables communication partners to decrypt data without the need to purchase a license of the TurboCrypt software package. No password information is stored in the selfextracting archive.

Securely delete plaintext files after encryption
By setting "Securely delete plaintext files after encryption" is set to the checked state, the selected original plaintext files are deleted by using one out of three available shredding methods: A simple 1-pass overwrite with random numbers, a three-pass shredding method that is used by the U.S. military or the ultra-secure Gutmann shredding algorithm which needs 35 passes to perform it's task.

After pressing the OK button, the following dialog box, which provides the user with the current progress, pops up:



As soon as the encryption process is finished, a new e-mail containing the zipped encrypted archive pops up as well. After sending the mail successfully, the encryption tool stops waiting.



-o-

Check for updates

Click at this button on the main window and the software will check for updates on the PMC Ciphers, Inc. website. If updates are available, they are downloaded and if you click at the "Install update" button, the new version is inflated and will be installed upon the next system boot.

! When clicking at this button, the software connects via HTTP protocol to the PMC Ciphers website www.pmc-ciphers.com.

-o-

Minimize to tray

When clicking at the minimize button on the upper right corner of the main window, the main window is minimized to the system tray on the right side of the task bar.

When right-clicking at the vault symbol in the task bar, a menu with several convenient options opens. As shown in the screenshot above, all mounted encrypted drives can be unmounted at a touch of a button.

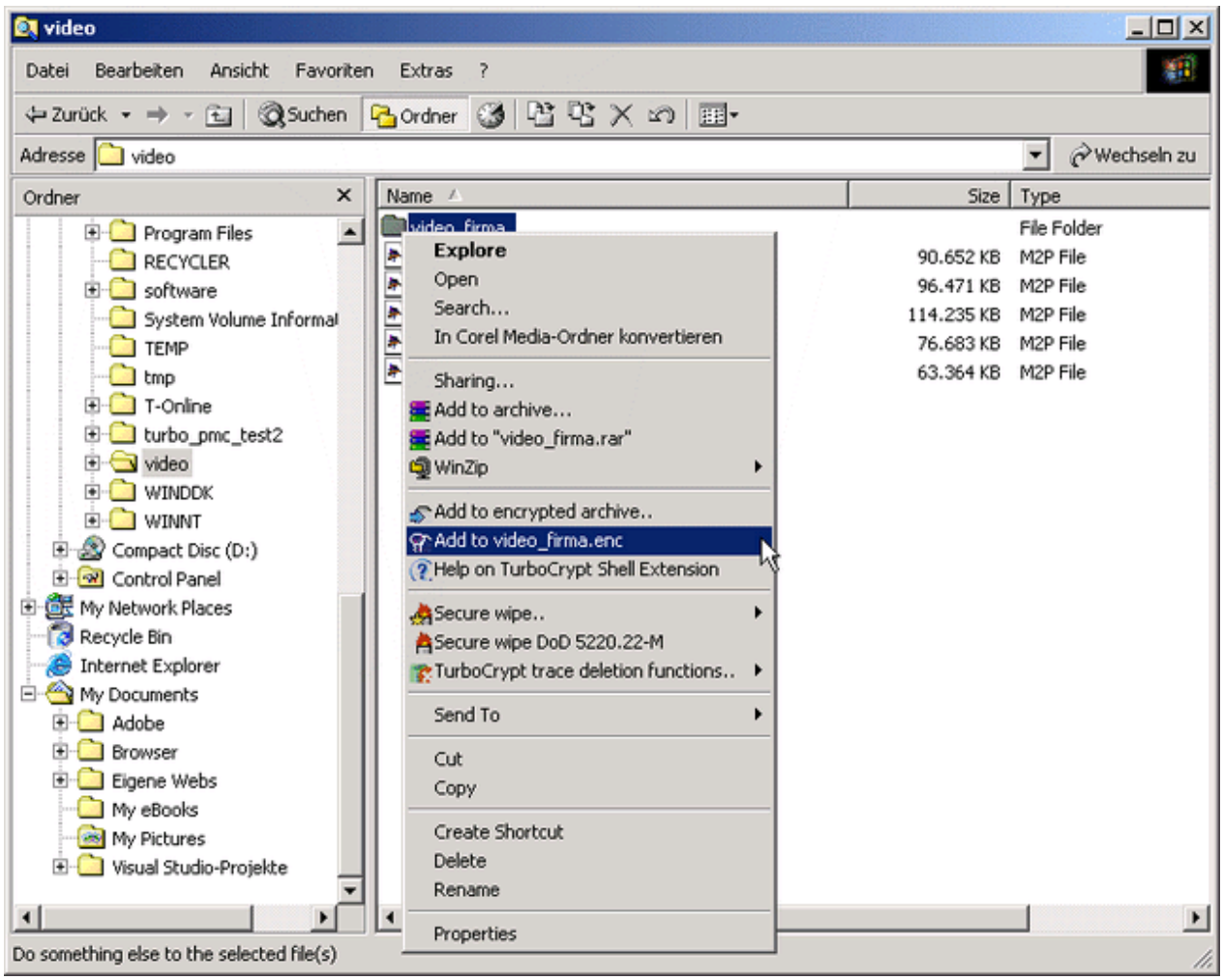
-o-

TurboCrypt Shell Extension functionality (installable versions only!)

TurboCrypt adds file encryption and secure wipe functionality to Windows Explorer. This functionality is not available for transportable TurboCrypt versions because it has to be installed on target systems.

The complete functionality of the file related functions of TurboCrypt is accessible through *clicking the right mouse button in Windows Explorer* as shown in the picture below. Files and folders can easily be encrypted, added to existing encrypted archives, or be securely wiped by using high-performance algorithms.

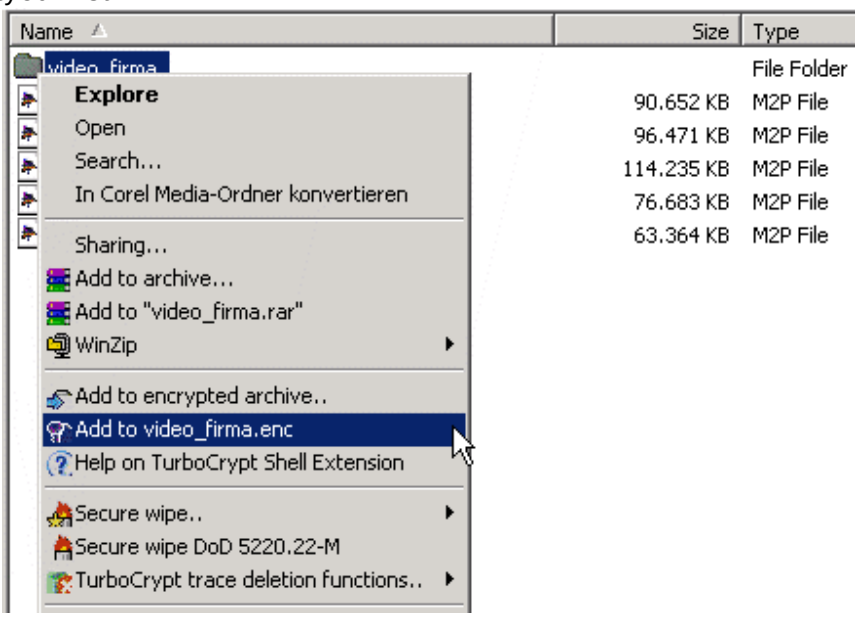
Windows Explorer context menu at right mouse click:



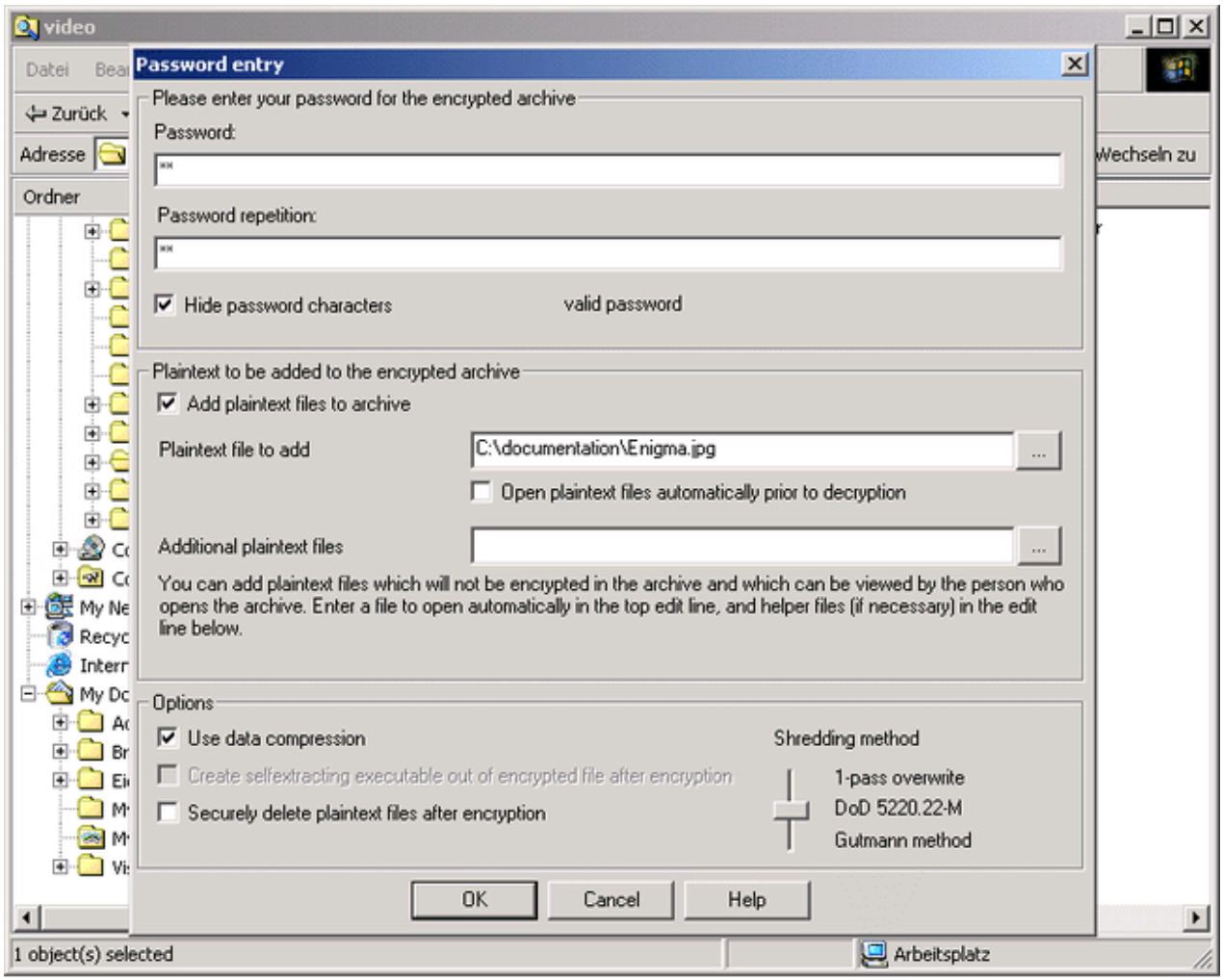
-o-

Add files to an encrypted archive

The selected file(s) and/or folder(s) are encrypted into a file archive. The directory tree structure is preserved within the archive. If selected, data which is to be encrypted can be compressed. It should be noted that data compression consumes a lot of CPU time. The user is first asked to specify a name for the archive that is to be created. For this purpose, a file dialog is displayed first.



The following password dialog pops up:



Password

Here you can enter the password to protect your encrypted volume. Please choose a long password like "Wa4X+g2#csdf89#2bDWXvtzks92m#fk6y10h". With the length of the password and the degree of uniqueness you indirectly choose the quality of the encryption: A short password like "Wa", corresponds with about 12 .. 16 bit encryption strength. Such simple passwords are very easy to crack!
The TurboCrypt Shell Extension maps all password entries to 256 bit long binary representations. Each character of your entry adds 6 ... 8 password bits. Consequently, after entering approximately 40 characters, no more effective password information is added.

If users seek protection against automatic cipher breaking software, they should ONLY rely on very long passwords to take full advantage of the ultra-strong 256 bit encryption used to encrypt file archives. Today, 128 bit keys are regarded as totally safe for the next 100 years. With every additional key bit, attack security increases by two (=> 129 bit keys are safe for at least 200 years, etc.).

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

Hide password characters
In order to hide the password characters that are typed in, an asterik is displayed instead of the actual characters. This option can be switched on and off by clicking at the checkbox.

Plaintext files

When encrypting commercial data, it might be useful to add a plaintext file which contains information on the type of data that is stored in the archive. This plaintext file remains plaintext in the archive and can be viewed by anyone. Files containing sound or video can also be added to the encrypted archive, thus e.g. providing users with a preview. HTML pages sometimes require additional files to be stored with the main html file. These files are added by clicking at the button next to the second line ("Additional plaintext files"). All the files which are added here should originate from the same directory as the main plaintext file ("Plaintext file to add"). When a user later wants to view the plaintext files, all available files from the archive are stored in the same temporary folder!

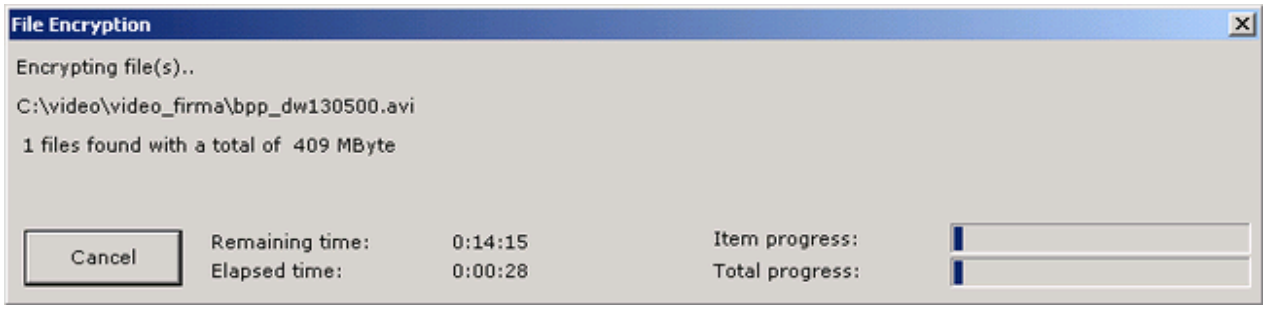
Options:

Use data compression
If the checkbox "Use data compression" is checked, the encryption engine will try to compress the plaintext prior to the actual encryption process. If the compression succeeds, the compressed representation of the data is encrypted and saved to the archive. Although highly optimized compression algorithms are implemented in the software, it should be noted that data compression consumes most of the CPU time.
Create selfextracting executable out of encrypted file after encryption
When setting this checkbox to the checked state, a program containing the decryption functions and the ciphertext file is additionally generated. The decryption functions add approximately 350kB to the file size. This function, which is only available for users who have purchased the software, enables communication partners to decrypt data without the need to purchase a license of the TurboCrypt software package. No password information is stored in the selfextracting archive.

! Prior to attaching selfextracting executable archives to e-mails, it is advisable to change the file name extension to something else than ".exe" (e.g. to ".e_x_e"). The reason for this is that many e-mail clients and firewalls block attachments that contain executables. Receivers of executable archives must change the file extension back to ".exe" in order to launch the selfextractor. Prior to launching the program, it is necessary to check the file for viruses!

Securely delete plaintext files after encryption
By setting "Securely delete plaintext files after encryption" is set to the checked state, the selected original plaintext files are deleted by using one out of three available shredding methods: A simple 1-pass overwrite with random numbers, a three-pass shredding method that is used by the U.S. military or the ultra-secure Gutmann shredding algorithm which needs 35 passes to perform it's task.

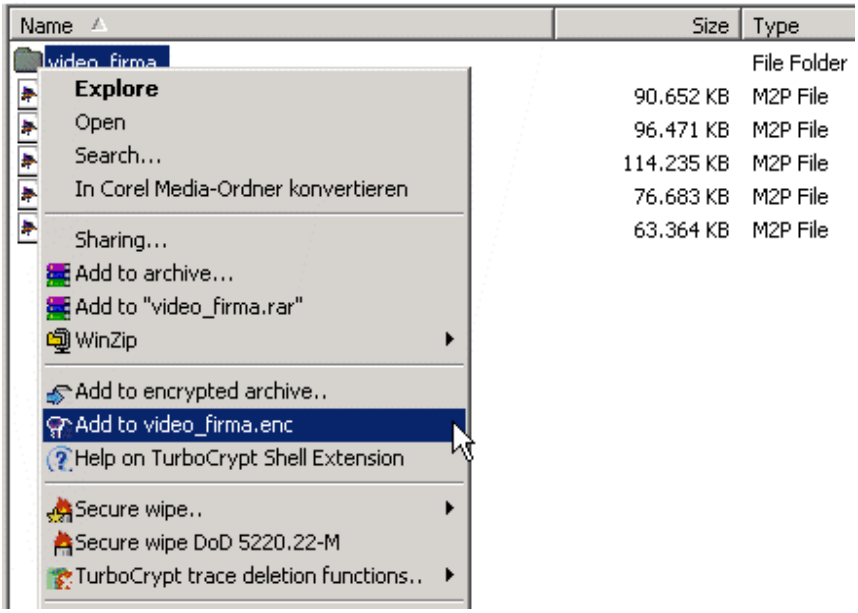
After pressing the OK button, the following dialog box, which provides the user with the current progress, pops up:



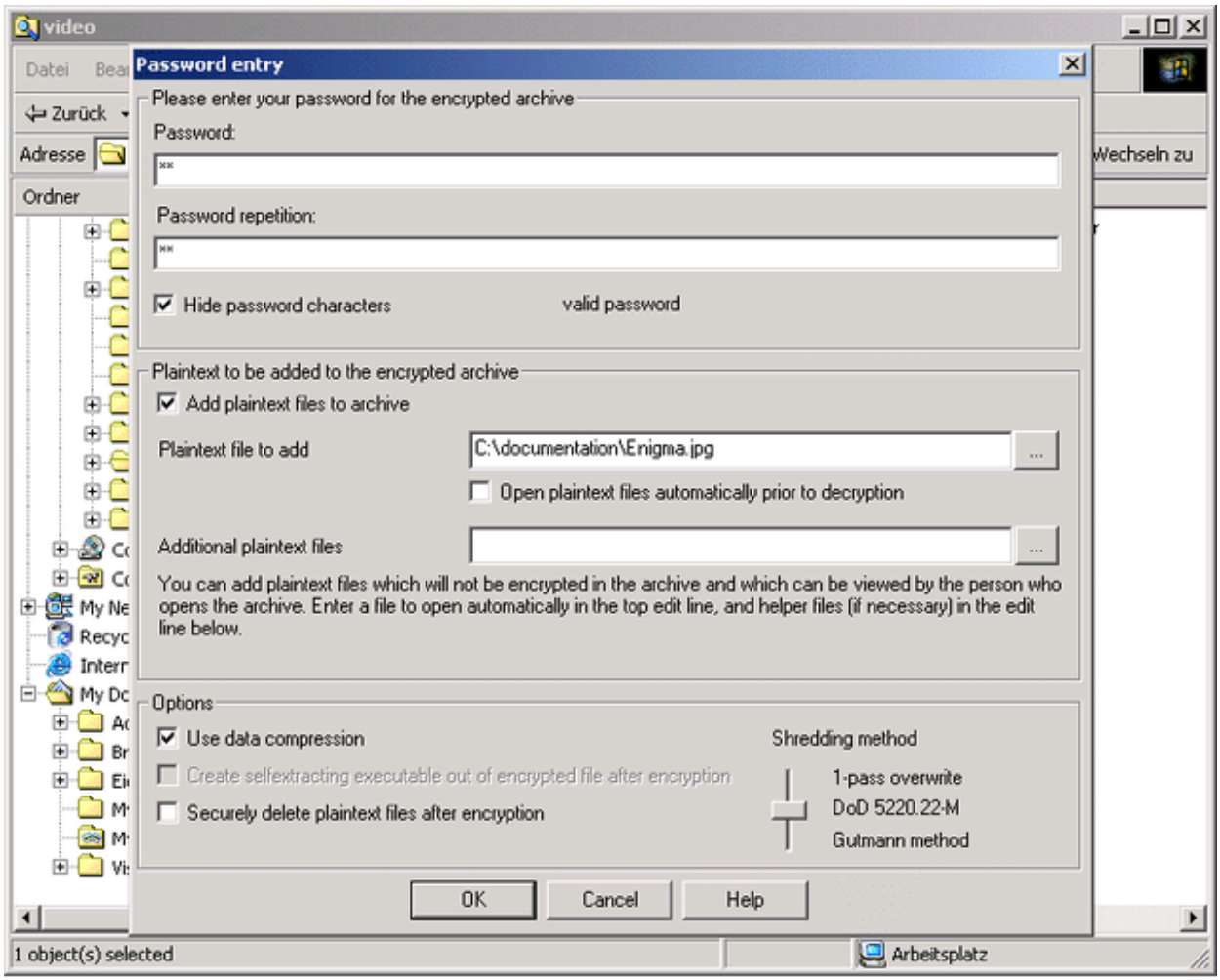
-o-

Add files to encrypted archive with proposed name

The selected file(s) and/or folder(s) are encrypted into a file archive. The directory tree structure is preserved within the archive. If selected, data which is to be encrypted can be compressed. It should be noted that data compression consumes a lot of CPU time. When choosing this menu item, the archive will be given the name that is proposed automatically by the software.



The following password dialog pops up:



Password
Here you can enter the password to protect your encrypted volume. Please choose a long password like "Wa4X+g2#csdf89#2bDWXvtzks92m#fk6y10h". With the length of the password and the degree of uniqueness you indirectly choose the quality of the encryption: A short password like "Wa", corresponds with about 12 .. 16 bit encryption strength. Such simple passwords are very easy to crack!
The TurboCrypt Shell Extension maps all password entries to 256 bit long binary representations. Each character of your entry adds 6 ... 8 password bits. Consequently, after entering approximately 40 characters, no more effective password information is added.

If users seek protection against automatic cipher breaking software, they should ONLY rely on very long passwords to take full advantage of the ultra-strong 256 bit encryption used to encrypt file archives. Today, 128 bit keys are regarded as totally safe for the next 100 years. With every additional key bit, attack security increases by two (=> 129 bit keys are safe for at least 200 years, etc.).

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

Hide password characters
In order to hide the password characters that are typed in, an asterik is displayed instead of the actual characters. This option can be switched on and off by clicking at the checkbox.

Plaintext files
When encrypting commercial data, it might be useful to add a plaintext file which contains

information on the type of data that is stored in the archive. This plaintext file remains plaintext in the archive and can be viewed by anyone. Files containing sound or video can also be added to the encrypted archive, thus e.g. providing users with a preview. HTML pages sometimes require additional files to be stored with the main html file. These files are added by clicking at the button next to the second line ("Additional plaintext files"). All the files which are added here should originate from the same directory as the main plaintext file ("Plaintext file to add"). When a user later wants to view the plaintext files, all available files from the archive are stored in the same temporary folder!

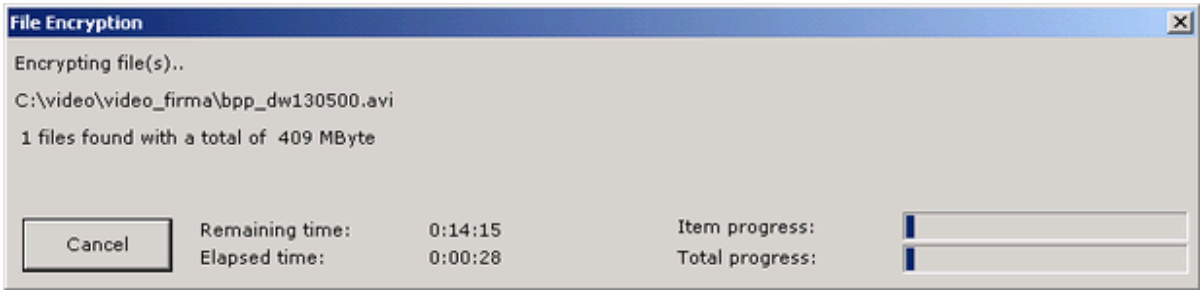
Options:

Use data compression
If the checkbox "Use data compression" is checked, the encryption engine will try to compress the plaintext prior to the actual encryption process. If the compression succeeds, the compressed representation of the data is encrypted and saved to the archive. Although highly optimized compression algorithms are implemented in the software, it should be noted that data compression consumes most of the CPU time.
Create selfextracting executable out of encrypted file after encryption
When setting this checkbox to the checked state, a program containing the decryption functions and the ciphertext file is additionally generated. The decryption functions add approximately 350kB to the file size. This function, which is only available for users who have purchased the software, enables communication partners to decrypt data without the need to purchase a license of the TurboCrypt software package. No password information is stored in the selfextracting archive.

! Prior to attaching selfextracting executable archives to e-mails, it is advisable to change the file name extension to something else than ".exe" (e.g. to ".e_x_e"). The reason for this is that many e-mail clients and firewalls block attachments that contain executables. Receivers of executable archives must change the file extension back to ".exe" in order to launch the selfextractor. Prior to launching the program, it is necessary to check the file for viruses!

Securely delete plaintext files after encryption
By setting "Securely delete plaintext files after encryption" is set to the checked state, the selected original plaintext files are deleted by using one out of three available shredding methods: A simple 1-pass overwrite with random numbers, a three-pass shredding method that is used by the U.S. military or the ultra-secure Gutmann shredding algorithm which needs 35 passes to perform it's task.

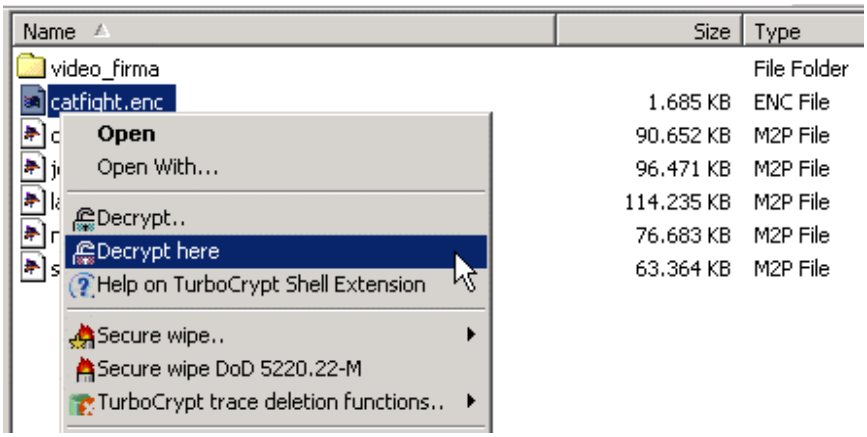
After pressing the OK button, the following dialog box, which provides the user with the current progress, pops up:



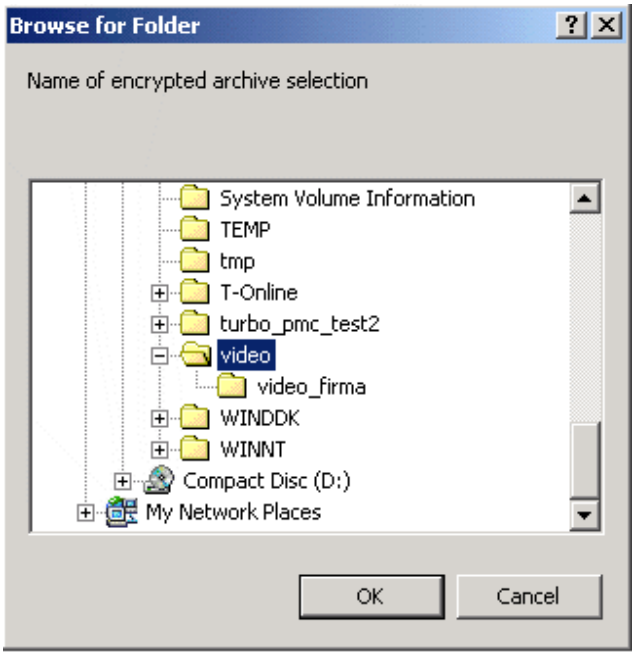
-o-

Decrypt

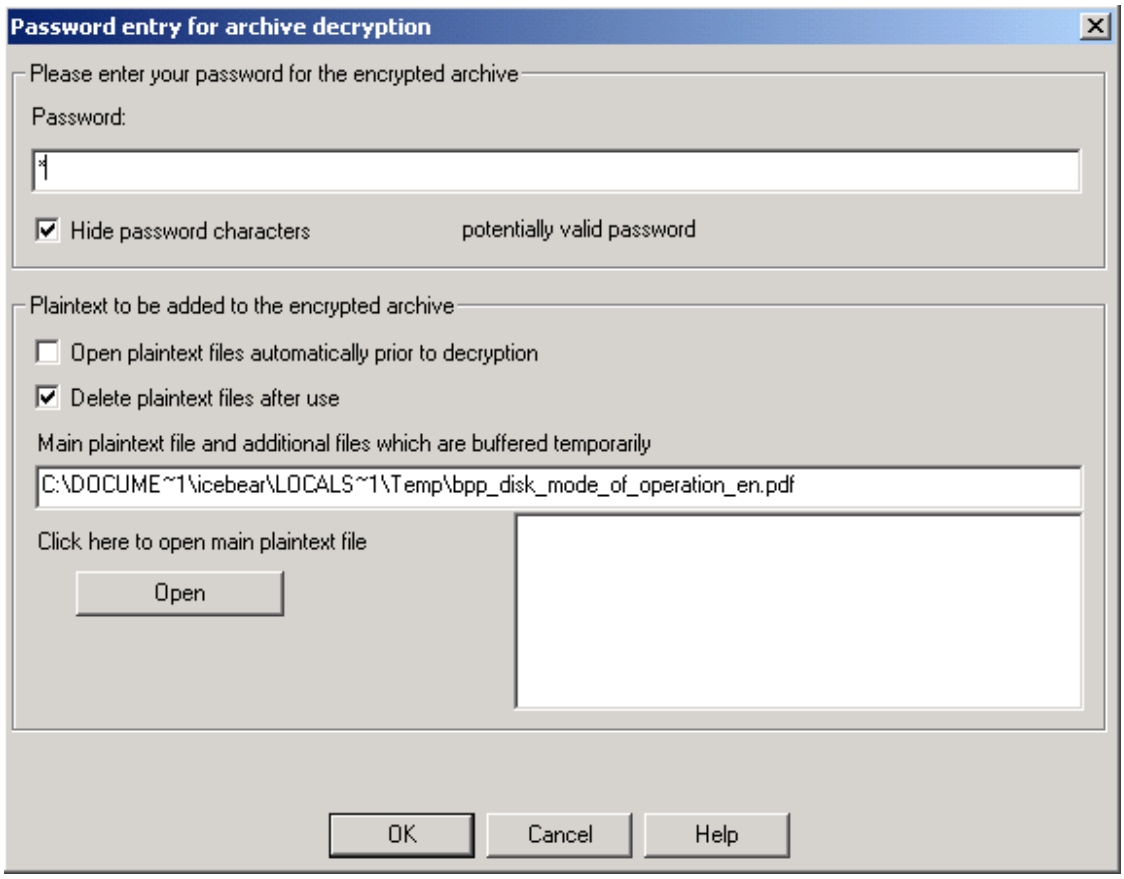
The selected encrypted archive is decrypted and inflated by selecting this function in Windows Explorer.



The directory tree structure of the archived files is restored beginning at the folder which is selected by the user in a window that pops up directly after launching this decryption function:



Once the desired base folder has been selected, the following password dialog pops up:



Password

In order to decrypt the archive, you must enter the password which was used to encrypt the archive in the first place. The archive does not contain any password information. Not even a checksum is stored with the archive during encryption. Consequently, the software can only check decrypted data for plausibility. If a wrong password has been entered, a window displaying an error pops up sooner or later. This feature makes TurboCrypt encrypted archives very hard to crack even if protected by short passwords as a lot of patience is required by unauthorized users.
If users seek protection against automatic cipher breaking software, they should ONLY rely on very long keys to take full advantage of the ultra-strong 256 bit encryption used to encrypt file archives.

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

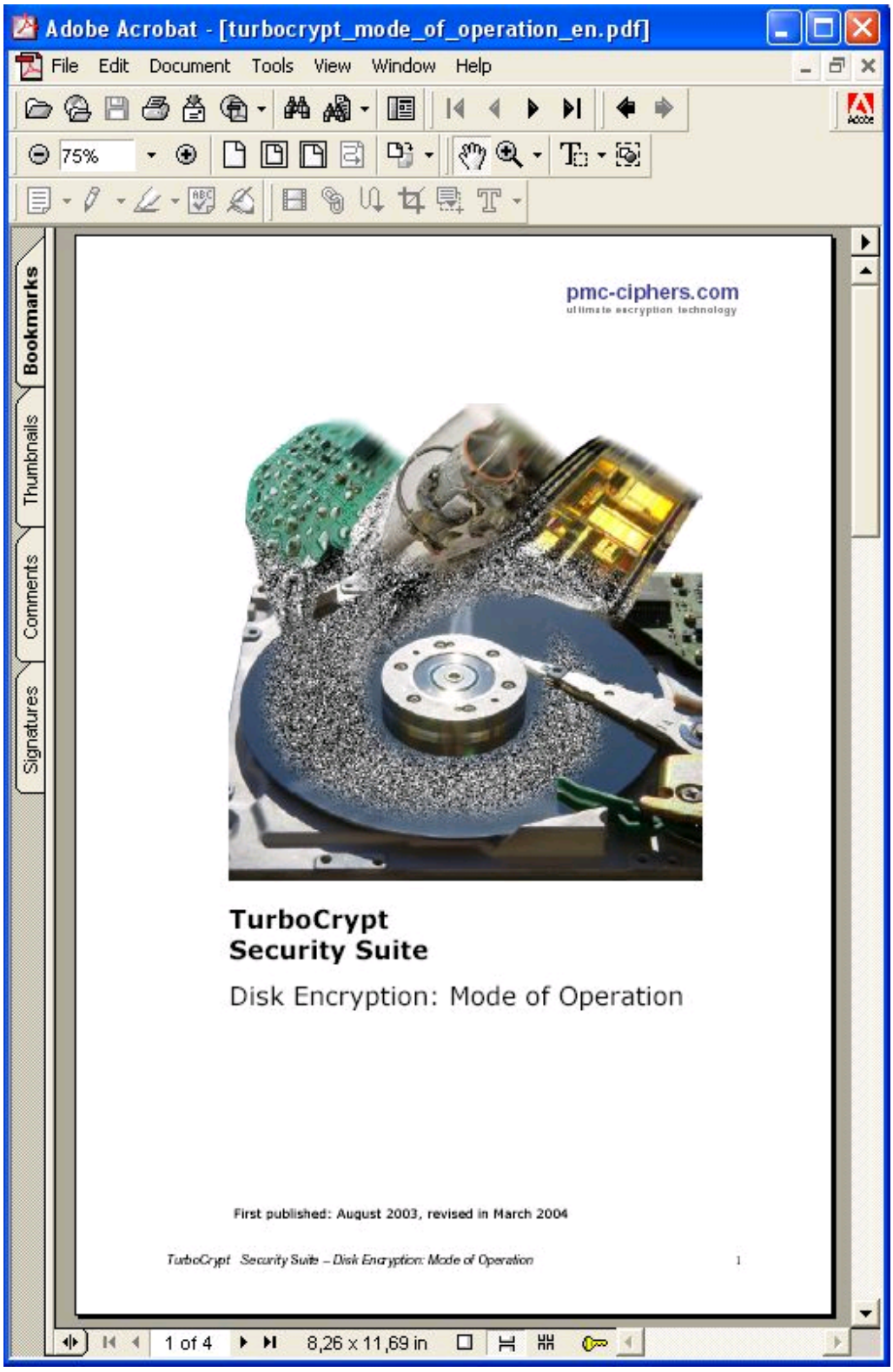
Plaintext files

Encrypted archives may contain one or more plaintext files in order to enable everybody to learn about the content of the archive, to provide copyright remarks, or to give users a preview of what is inside the encrypted archive.
The main plaintext file can either be viewed by clicking the "Open" button or it can be automatically opened by setting the appropriate checkbox to the checked state.

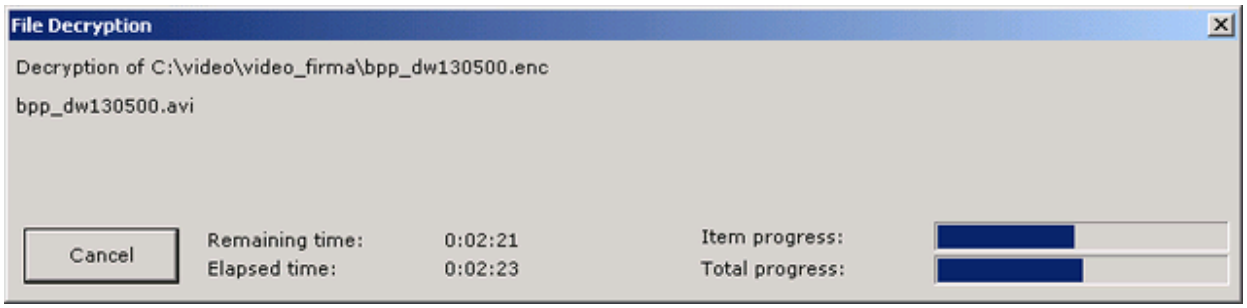
TurboCrypt copies the plaintext files to the temporary directory specified by the operating system. The files are deleted immediately after pressing the "OK" or the "Cancel" button if the checkbox "delete temporary files after use" is set.

As an example, an archive containing a detailed description of secret internals of TurboCrypt

may contain this publically known PDF document as plaintext file in order to let people know what is inside the archive:



If the user enters the correct password, archive decryption is started by pressing the OK button.
The following dialog box, which provides the user with the current progress, pops up:

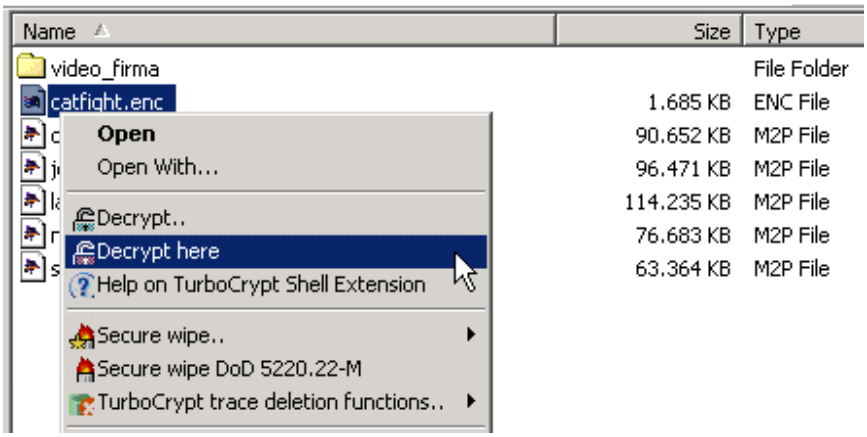


In case the wrong password has been entered, the software will either display an error message or it will try to analyze the archive for some time and will then display an error message.

-o-

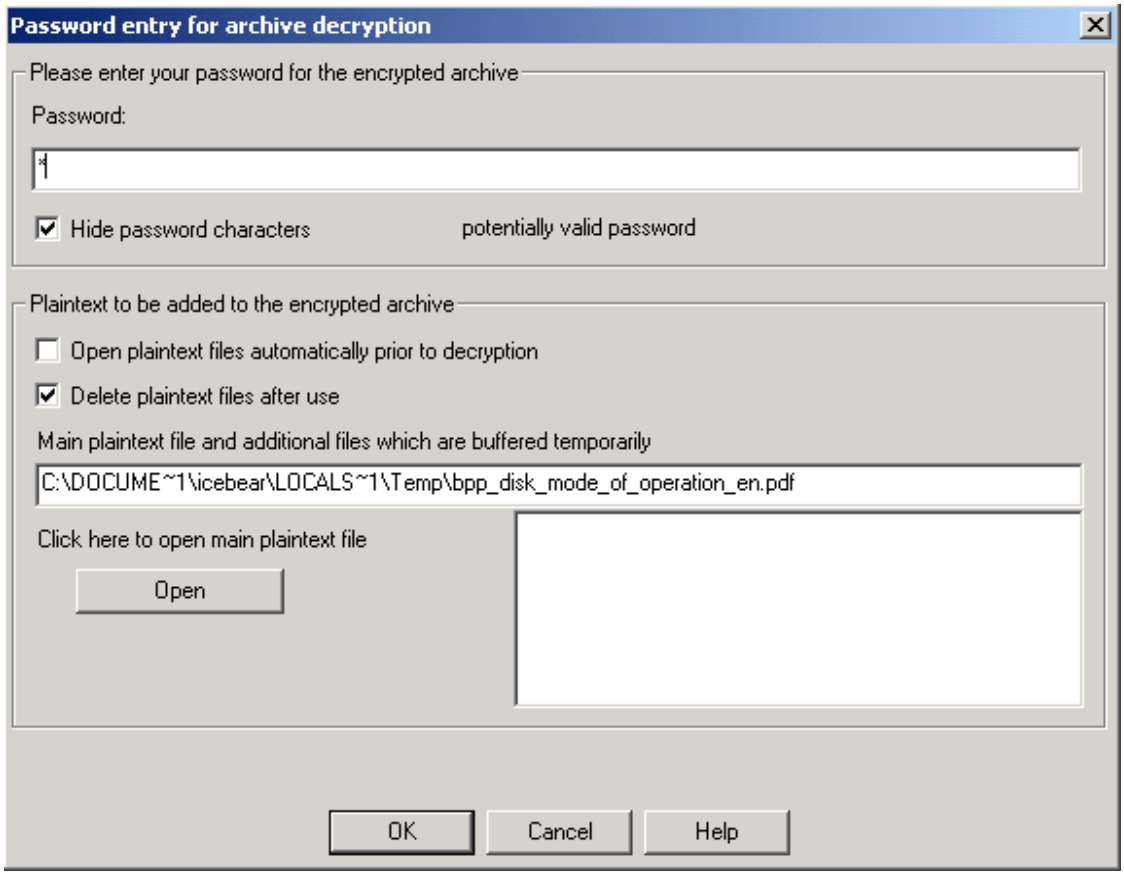
Decrypt here

The selected encrypted archive is decrypted and inflated by selecting this function in Windows Explorer.



The directory tree structure of the archived files is restored beginning at the currently selected folder.

Once the desired base folder has been selected, the following password dialog pops up:



Password

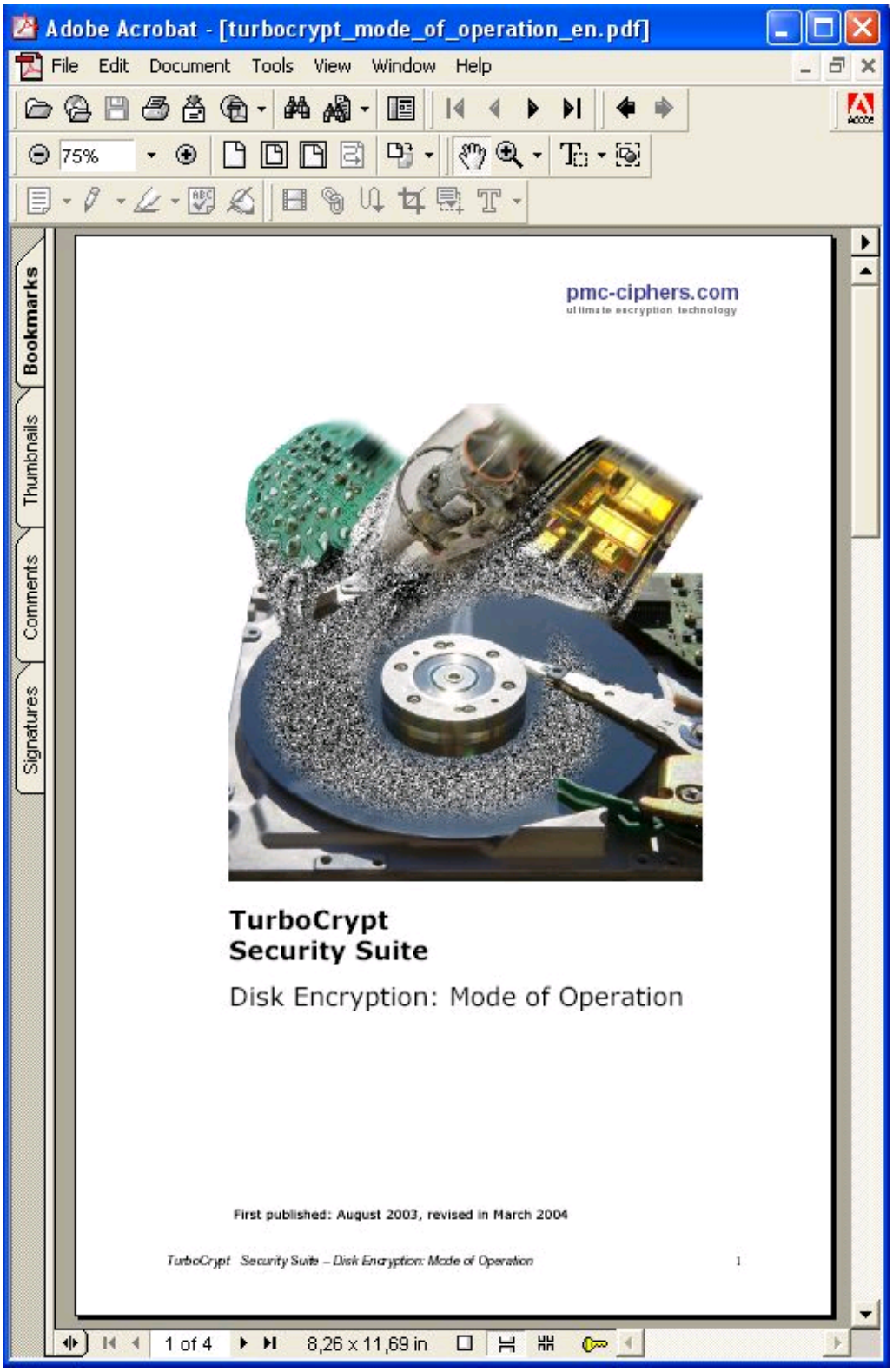
In order to decrypt the archive, you must enter the password which was used to encrypt the archive in the first place. The archive does not contain any password information. Not even a checksum is stored with the archive during encryption. Consequently, the software can only check decrypted data for plausibility. If a wrong password has been entered, a window displaying an error pops up sooner or later. This feature makes TurboCrypt encrypted archives very hard to crack even if protected by short passwords as a lot of patience is required by unauthorized users.
If users seek protection against automatic cipher breaking software, they should ONLY rely on very long keys to take full advantage of the ultra-strong 256 bit encryption used to encrypt file archives.

! Please make sure that nobody else but you knows your password because it is the only way to access your secured data.
If you forget your password, you will never be able to access your data again.

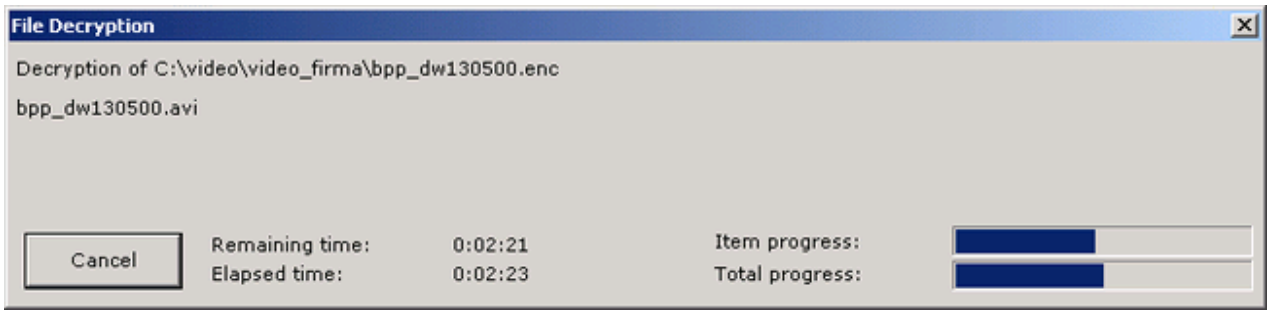
Plaintext files
Encrypted archives may contain one or more plaintext files in order to enable everybody to learn about the content of the archive, to provide copyright remarks, or to give users a preview of what is inside the encrypted archive.
The main plaintext file can either be viewed by clicking the "Open" button or it can be automatically opened by setting the appropriate checkbox to the checked state.

TurboCrypt copies the plaintext files to the temporary directory specified by the operating system. The files are deleted immediately after pressing the "OK" or the "Cancel" button if the checkbox "delete temporary files after use" is set.

As an example, an archive containing a detailed description of secret internals of TurboCrypt may contain this publically known PDF document as plaintext file in order to let people know what is inside the archive:



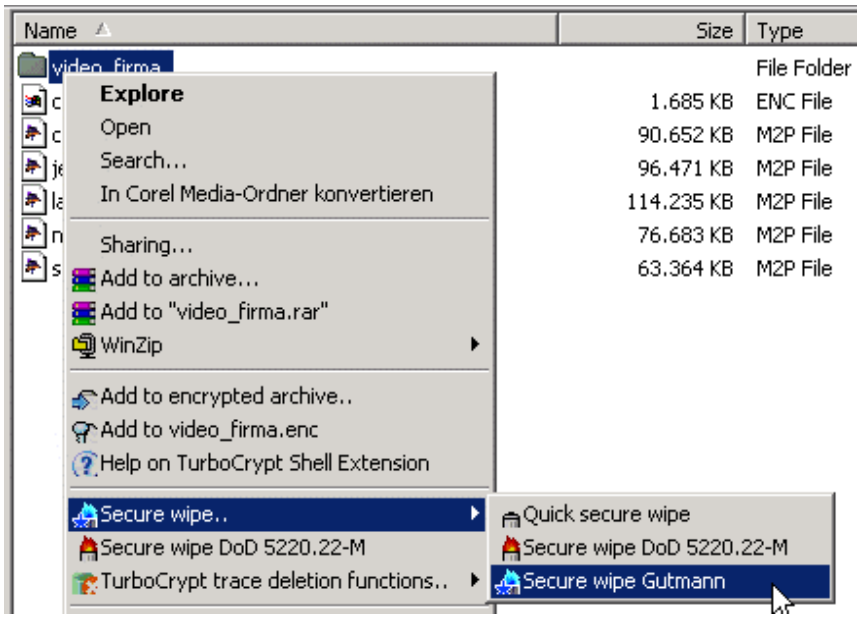
If the user enters the correct password, archive decryption is started by pressing the OK button.
The following dialog box, which provides the user with the current progress, pops up:



In case the wrong password has been entered, the software will either display an error message or it will try to analyze the archive for some time and will then display an error message.

-o-

Secure wipe (file and folder shredding)



The selected file(s) and/or folder(s) are overwritten using three different algorithms and they are subsequently deleted:

Wipe: The selected items are overwritten with real random numbers and after finishing this process, they are deleted.

This method is not approved by the DoD for sanitizing media that contain top secret information. This method does not take into account, that the hard disk head does not always fly over the center of the track.

Wipe DoD 5220.22-M: The selected items are overwritten three times prior to deleting them according to the standard 5220.22-M of the U.S. Department of Defense:
Step 1: Overwrite all addressable locations with a character
Step 2: Overwrite all addressable locations with with the complement of the previously written chracter
Step 3: Overwrite with a random character
Step 4: Verify the data which was previously written to the writable medium.

It should be noted that this method is not approved by the DoD for sanitizing media that contain top secret information

Wipe Gutmann: The selected items are overwritten 35 times prior to deleting them according to Peter Gutmann's method which was proposed in 1996 in his paper "Secure Deletion of Data from Magnetic and Solid-State Memory".

	Overwrite Data
Pass No.	**Data Written (Binary/Hexadecimal)**

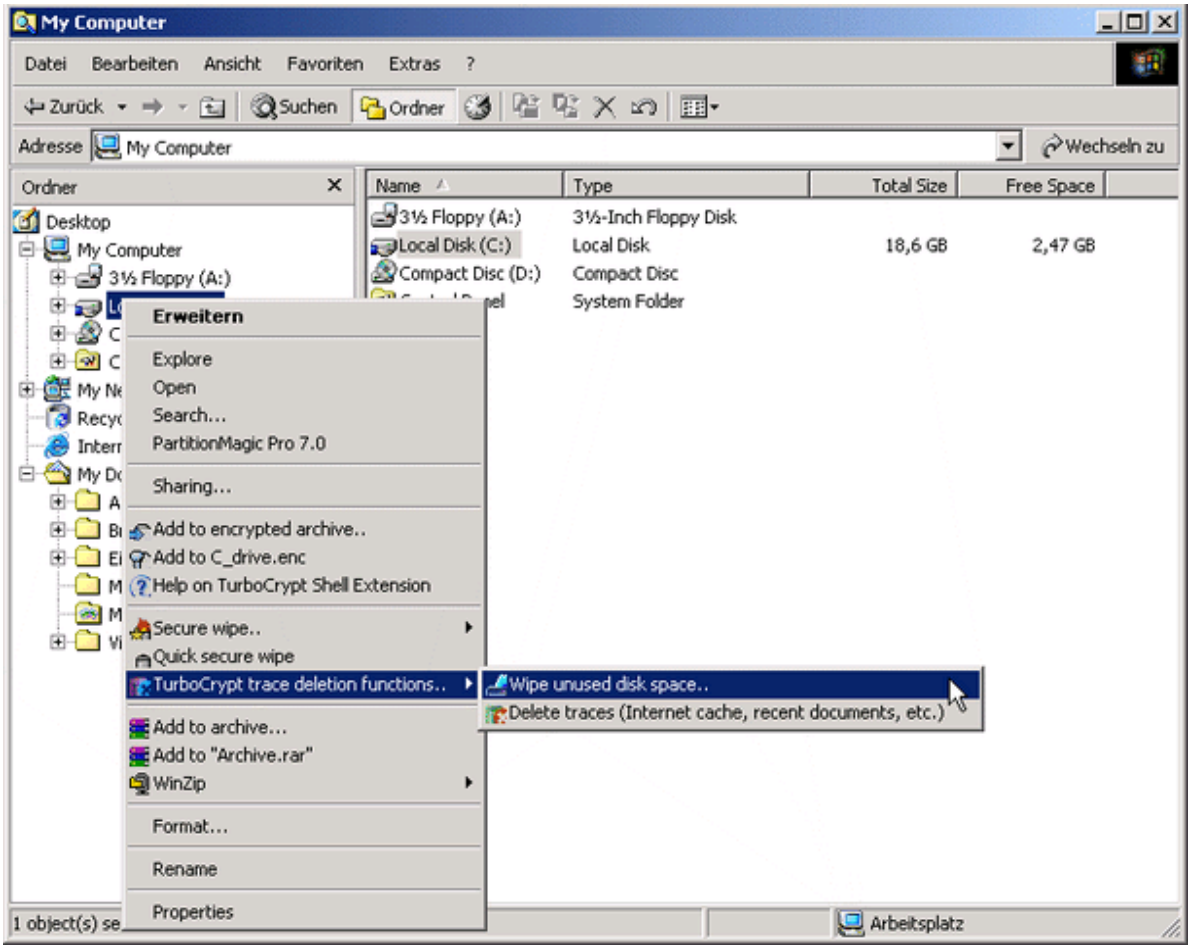
1	Random numbers	
2	Random numbers	
3	Random numbers	
4	Random numbers	
5	01010101 01010101 01010101	0x55
6	10101010 10101010 10101010	0xAA
7	10010010 01001001 00100100	0x92 0x49 0x24
8	01001001 00100100 10010010	0x49 0x24 0x92
9	00100100 10010010 01001001	0x24 0x92 0x49
10	00000000 00000000 00000000	0x00
11	00010001 00010001 00010001	0x11
12	00100010 00100010 00100010	0x22
13	00110011 00110011 00110011	0x33
14	01000100 01000100 01000100	0x44
15	01010101 01010101 01010101	0x55
16	01100110 01100110 01100110	0x66
17	01110111 01110111 01110111	0x77
18	10001000 10001000 10001000	0x88
19	10011001 10011001 10011001	0x99
20	10101010 10101010 10101010	0xAA
21	10111011 10111011 10111011	0xBB
22	11001100 11001100 11001100	0xCC
23	11011101 11011101 11011101	0xDD
24	11101110 11101110 11101110	0xEE
25	11111111 11111111 11111111	0xFF
26	10010010 01001001 00100100	0x92 0x49 0x24
27	01001001 00100100 10010010	0x49 0x24 0x92
28	00100100 10010010 01001001	0x24 0x92 0x49
29	01101101 10110110 11011011	0x6D 0xB6 0xDB
30	10110110 11011011 01101101	0xB6 0xDB 0x6D
31	11011011 01101101 10110110	0xDB 0x6D 0xB6
32	Random numbers	
33	Random numbers	
34	Random numbers	
35	Random numbers	

The implemented algorithm is regarded as very secure throughout the industry. Although, when using this method, the consumption of a lot of processing time must be taken into account.

-o-

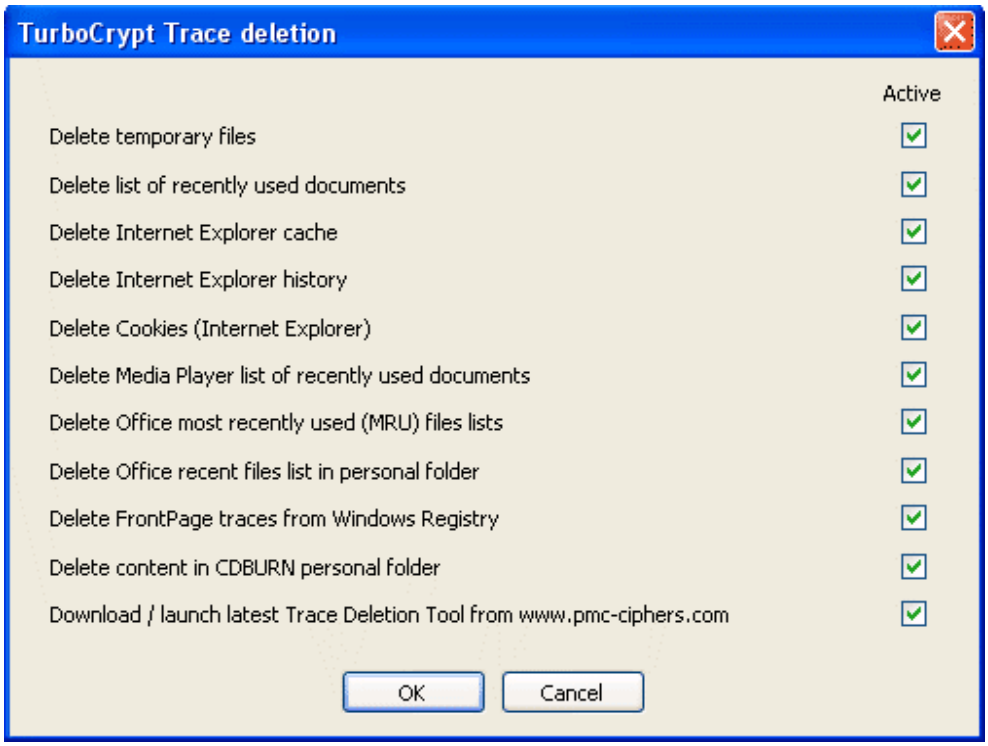
Wipe unused disk space

When launching this function, a window pops up in which the user can choose on which writable medium he wants to wipe unused hard disk space. This function writes random numbers to the remaining storage capacity of the selected drive.
As the free capacity of modern hard disks is usually in the range of several 100 to 500 gigabytes, the implemented one-pass algorithm is optimized for speed.



-o-

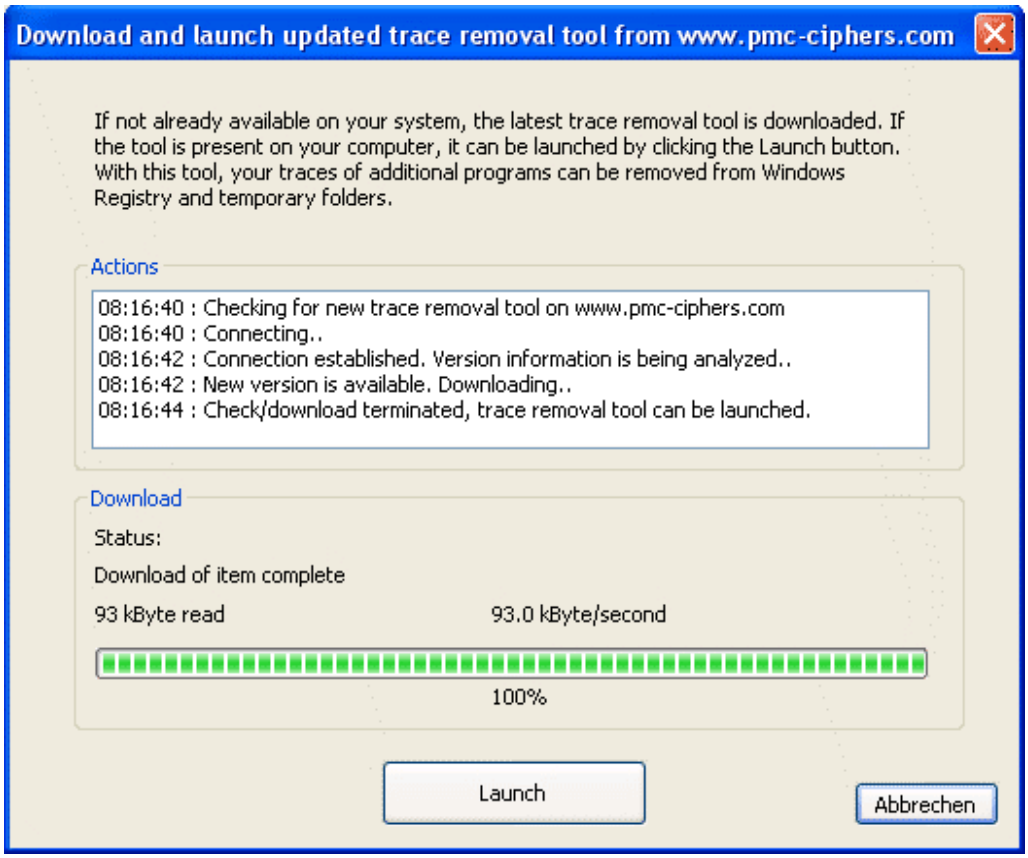
Trace deletion



With this function, a number of items containing data about your habits and your work are deleted:
Internet browser remains like cookies, history lists, etc.

- Temporary files
- List of recently used documents, Media Player- and Office MRU lists, Office recent files in personal folder
- FrontPage registry traces
- CDBURN personal folder

Additionally, a function for automatic download and launch of the most up-to-date trace deletion extension utility is provided. Currently this tool cleans traces left by use of the RealPlayer.



By clicking at the Launch button, the updated trace removal extension utility is launched.

-o-

Registering TurboCrypt

The unregistered version of TurboCrypt is a full version and can be used for evaluation purposes, as well as at home.
For these applications, the unregistered TurboCrypt Encryption Suite is freeware and can thus be copied, distributed and used freely.

Benefits when registering TurboCrypt
1, 5 or an unlimited number of encrypted volumes
Up to 2Terabyte (2000 Gigabyte) volume size of encrypted drives
File and folder encryption including selfextraction capability
File and folder encryption: Fast and highly efficient compression that outperforms WinZIP
Encrypted NTFS raw devices
Email attachment encryption

You can purchase your licence on our web site at:
http://www.turbocrypt.com

Alternatively we accept orders by phone or e-mail:
Phone: (716) 566 2780 (outside the U.S.: +1 716 566 2780)
E-mail: sales@turbocrypt.com

Benefits of TurboCrypt
Unbreakable: Full strength of the Polymorphic 512 bit Cipher (AES version: 2x 256 bit AES Rijndael)
256MB volume size of encrypted volume **=> Register and use 5 or more volumes each with up to 2000GB volume size**
AES version: Encryption algorithm is fully FIPS-197 compliant
Secure wipe of files, folders and unused disk space
Deletion of cookies, internet history list, Internet Explorer cache, recently used document list, temporary files, Media Player recently used files list, Office MRU lists, Office recent files in personal folder, FrontPage registry traces, CDBURN personal folder, etc..
Update of trace removal extension from our website
Very reliable operation

File hosted volumes can be shared on a network
Chkdsk / defrag capability
TurboCrypt among 33 best tools in the world according to PC Welt magazine

System requirements
Windows XP, Windows Server 2003 or Windows 2000
Pentium processor or compatible (e.g. PIII, PIV, Duron, Athlon, Athlon XP, etc.)
10MB free hard disk space for the installation and some free disk space for your encrypted data

Registation procedure:

You can access this dialog box through the main menu, but it also pops up at program start if the software is still unregistered.

1.) Order one of the available license options online or by telephone, then send us the software key in the dialog box as displayed on your computer's screen (please see the screenshot above). The example above shows the following software key: A01-A96FA84B-8ECFD20F-70. You can copy-paste your software key into your e-mail to sales@turbocrypt.com

2.) After us sending your registration key to you, please enter it in the edit line of the dialog box and click OK.
After relaunching TurboCrypt, you will be able to create large encrypted virtual volumes and to share encrypted information via Internet using the file and folder encryption functionality of TurboCrypt.

-o-

Index

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- V -

- W -

© 1999-2005 PMC-Ciphers, Inc, All Rights Reserved