kiCrypt DLL Demo Kenneth Ives kenaso@tx.rr.com

I am open to ways to improve this application, please email me.

Visual Basic 6.0 with Service Pack 6 runtime files required.
To obtain required files (VBRun60sp6.exe):
http://www.microsoft.com/downloads/details.aspx?FamilyId=7B9BA261-7A9C-43E7-9117-F6730
77FFB3C

VBRun60sp6.exe installs Visual Basic 6.0 SP6 run-time files.
http://support.microsoft.com/kb/290887

This software has been tested on Windows XP SP3 64-bit through Windows 10.
Windows XP 32-bit, 9x, 2000 and NT4 are no longer supported.

All algorithms, with the exception of Base64, can process files
in excess of 2gb.

=
*** WARNING *** WARNING *** WARNING *** WARNING *** WARNING *** WARNING ***
=

You acknowledge that this software is subject to the export control laws and regulations of the United States ("U.S.") and agree to abide by those laws and regulations. Under U.S. law, this software may not be downloaded or otherwise exported, reexported, or transferred to restricted countries, restricted end-users, or for restricted end-uses. The U.S. currently has embargo restrictions against Cuba, Iran, Iraq, Libya, North Korea, Sudan, and Syria. The lists of restricted end-users are maintained on the U.S. Commerce Department's Denied Persons List, the Commerce Department's Entity List, the Commerce Department's List of Unverified Persons, and the U.S. Treasury Department's List of Specially Designated Nationals and Blocked Persons. In addition, this software may not be downloaded or otherwise exported, reexported, or transferred to an end-user engaged in activities related to weapons of mass destruction.

=

REFERENCE:

NIST (National Institute of Standards and Technology)
FIPS (Federal Information Processing Standards Publication)
SP (Special Publications)
http://csrc.nist.gov/publications/PubsFIPS.html

FIPS 180-2 (Federal Information Processing Standards Publication)
dated 1-Aug-2002, with Change Notice 1, dated 25-Feb-2004
http://csrc.nist.gov/publications/fips/fips180-2/FIPS180-2_changenotice.pdf

FIPS 180-3 (Federal Information Processing Standards Publication)
dated Oct-2008 (supercedes FIPS 180-2)
http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

FIPS 180-4 (Federal Information Processing Standards Publication)
dated Mar-2012 (Supercedes FIPS-180-3)
http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

Examples of SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 are available at
http://csrc.nist.gov/groups/ST/toolkit/examples.html

Guidelines for Media Sanitization (SP800-88)
http://csrc.nist.gov/publications/nistpubs/800-88/NISTSP800-88_rev1.pdf

Examples of the implementation of the secure hash algorithms
SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and
SHA-512/256, can be found at:
http://csrc.nist.gov/groups/ST/toolkit/examples.html
http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/SHA2_Additional.pdf

Aaron Gifford's additional test vectors
http://www.adg.us/computers/sha.html

Feb-2005: SHA-1 has been compromised. Recommended that
you do not use for password or document authentication.
http://www.schneier.com/blog/archives/2005/02/sha1_broken.html
http://csrc.nist.gov/groups/ST/toolkit/documents/shs/NISTHashComments-final.pdf

March 15, 2006: The SHA-2 family of hash functions
(i.e., SHA-224, SHA-256, SHA-384 and SHA-512) may be used
by Federal agencies for all applications using secure hash
algorithms. Federal agencies should stop using SHA-1 for
digital signatures, digital time stamping and other
applications that require collision resistance as soon as
practical, and must use the SHA-2 family of hash functions
for these applications after 2010. After 2010, Federal
agencies may use SHA-1 only for the following applications:
    - hash-based message authentication codes (HMACs)
    - key derivation functions (KDFs)
    - random number generators (RNGs)
Regardless of use, NIST encourages application and protocol
designers to use the SHA-2 family of hash functions for all
new applications and protocols.
http://csrc.nist.gov/groups/ST/hash/policy.html

Export Control: Certain cryptographic devices and technical
data regarding them are subject to Federal export controls.
Exports of cryptographic modules implementing this standard
and technical data regarding them must comply with these
Federal regulations and be licensed by the Bureau of Export
Administration of the U.S. Department of Commerce.
Information about export regulations is available at:
http://www.bis.doc.gov/index.htm

SHA-2 support on MS Windows
Paraphrasing: Regarding SHA-224 support, SHA-224 offers less security
than SHA-256 but takes the same amount of resources. Also SHA-224 is
not generally used by protocols and applications. The NSA's (National
Security Agency) Suite B standards also does not include it. Microsoft
has no plans to add it to future versions of their Cryptographic
Service Providers (CSP).

http://blogs.msdn.com/b/alejacma/archive/2009/01/23/sha-2-support-on-windows-xp.aspx

****

How to use:

For a simple example, execute the SHA_Demo application. The demo converts
the data to a byte array prior to passing it to the DLL to be hashed.

[STRING DATA]
Convert string data to byte array prior to passing to the HashString function.

  Ex: abytData() = StrConv("abc", vbFromUnicode)

[FILE DATA]

Just the path and filename are passed in the byte array. Convert the
path\filename data to byte array prior to passing to the HashFile function.
The HashFile routine will open and read the file into an internal byte array.

```
  Ex:  abytData() = StrConv("C:\Files\Test Folder\Testfile.txt", vbFromUnicode)
```

Both will create a hashed output string based on file data input.

Test data provided to test either hash or cipher:

```
    TestPhrase.txt  ASCII text phrase    (Copy & paste phrase for string test)
    API32.txt       ASCII text file      1.24 MB (1,308,504 bytes)
    TestFile.txt    ASCII text file      5.23 KB (5,360 bytes)

  Binary test files:
    kB_32.dat       32,768 binary zeros
    OneMil_a.dat    One million letter "a"    (As per FIPS 180-2)
    OneMil_0.dat    One million binary zeros  (As per FIPS 180-3)
```

****

Size definitions used by various disk manufacturers

Bit	0 or 1
Nibble	4 Bits
Byte	8 Bits
Kibibit	1,024 bits
Kilobit	1,000 bits
Kibibyte	1,024 bytes
Kilobyte	1,000 bytes
Mebibit	1,048,576 bits
Megabit	1,000,000 bits
Mebibyte	1,048,576 bytes
Megabyte	1,000,000 bytes
Gibibit	1,073,741,824 bits
Gigabit	1,000,000,000 bits
Gibibyte	1,073,741,824 bytes
Gigabyte	1,000,000,000 bytes
Tebibit	1,099,511,627,776 bits
Terabit	1,000,000,000,000 bits
Tebibyte	1,099,511,627,776 bytes
Terabyte	1,000,000,000,000 bytes
Pebibit	1,125,899,906,842,624 bits
Petabit	1,000,000,000,000,000 bits
Pebibyte	1,125,899,906,842,624 bytes
Petabyte	1,000,000,000,000,000 bytes

In 1998 the IEC changed it's measurements so that what you consider a
gigabyte (1024mb or 2^30) was renamed a gibibyte. A gibibyte is now
formally recognised as 1000mb, even though no operating system (like
windows) use this definition.

Hard drives use the term "gigabytes" which would be what the 40GB
stands for. But, Microsoft uses a different way to measure gigabytes
(they're actually gibibytes) so that's why you "lost" some 3.8GB from
your hard drive. It happens to everyone. It's just the conflict of
two different numbering systems.

The difference between those two numbering systems is seven percent.
So, when you buy a 40GB drive, Windows sees it as 37.2 GB because:

        40 GB - 7% = 37.2 GB

```
********************************
** PASSWORDS
********************************
```

Currently there is a minimum and maximum length of the password the user may enter. This can be changed in the kiCrypt DLL modCommon.bas module. In the declarations section, locate these two constants and make the desired change. Be sure to recompile the DLL and the demo application.

```
PWD_LENGTH_MIN = 8
PWD_LENGTH_MAX = 50
```

If no hash algorithm is selected then the default will be SHA-256. If SHA2 capability is not available, the default will be SHA-1.

****

Project: ArcFour Encryption/decryption

Description: ArcFour is a stream cipher symmetric key algorithm. Very similar to RC4 that was developed in 1987 by Ronald Rivest and kept as a trade secret by RSA Data Security. On September 9, 1994, the RC4 algorithm was anonymously posted on the Internet on the Cyperpunks' anonymous remailers list.

The name "RC4" is trademarked. The current status seems to be that "unofficial" implementations are legal, but cannot use the RC4 name. RC4 is often referred to as "ARCFOUR" or "ARC4" (meaning Alleged RC4, because RSA has never officially released the algorithm). This is to avoid possible trademark problems.

ArcFour uses a variable length key from 1 to 256 bytes to initialize a 256-byte state table. The state table is used for subsequent generation of pseudo-random bytes and then to generate a pseudo-random stream which is XORed with the plaintext to give the ciphertext. Each element in the state table is Swapped at least once.

The ArcFour key is often limited to 40 bits, because of export restrictions but it is sometimes used as a 128 bit key. It has the capability of using keys between 1 and 2048 bits. ArcFour is used in many commercial software packages such as Lotus Notes and Oracle Secure SQL. It is also part of the Cellular Specification.

The ArcFour algorithm works in two phases, key setup and ciphering. Key setup is the first and most difficult phase of this algorithm. During a N-bit key setup (N being your key length), the encryption key is used to generate an encrypting variable using two arrays, state and key, and N-number of mixing operations. These mixing operations consist of swapping bytes, modulo operations, and other formulas. A modulo operation is the process of yielding a remainder from division.

Once the encrypting variable is produced from the key setup, it enters the ciphering phase, where it is XORed with the plain text message to create and encrypted message. XOR is the logical operation of comparing two binary bits. If the bits are different, the result is 1. If the bits are the same, the result is 0. Once the receiver gets the encrypted message, he decrypts it by XORing the encrypted message with the same encrypting variable.

```
                ********************************************************
SPECIAL NOTE:   The encryption process has been enhanced.

                - Encryption process can be performed multiple times.
                  See property EncryptRounds().

                - Password key length is expanded to meet and exceed AES
                  standards.  Key lengths are 128 to 448 in 32 bit
                  increments and 448 to 1024 in 64 bit increments.  See
                  property KeyLength().

                - S-Box data will be mixed multiple times.
                ********************************************************

Reference:      RC4 Encryption Algorithm
                www.vocal.com/RC4.pdf

                RC4 from Wikipedia, the free encyclopedia
                http://en.wikipedia.org/wiki/RC4_(cipher)

                Ronald Rivest Home Page
                http://theory.lcs.mit.edu/~rivest/

                An Introduction to Using Keys in Cryptography
                By DI Management Services Pty Ltd, Sydney Australia
                http://www.di-mgt.com.au/cryptokeys.html

*************************************************************************

Project:        Base64 Encryption/Decryption
                (Originally named clsRadix64, Radix 64 encoding/decoding functions)

Description:    RFC 1341 - MIME (Multipurpose Internet Mail Extensions)
                Mechanisms for Specifying and Describing the Format of
                Internet Message Bodies
                http://www.faqs.org/ (highly recommended)

                What is BASE64?  A method of encoding binary data within
                text. If you'll remember, binary data is a full 8-bits per
                byte, whereas text uses a little more than 6 bits per byte.
                A 6-bit number has 64 combinations, hence the term "BASE64".

                The way it works is that every three 8-bit bytes are stored
                in four 6-bit characters, where the characters are in the
                range [A-Z][a-z][0-9][+/]. (Count 'em up; that's 64 total
                characters).  Since this doesn't exactly line up, pad
                characters of [=] are used at the very end.

                Where Base64 Is used?  HTTP "Basic" Authentication and PGP
                signatures And keys.  One of many ways of encoding e-mail.

                Borenstein & Freed                              [Page 16]

                RFC 1341MIME: Multipurpose Internet Mail ExtensionsJune 1992

                5.2  Base64 Content-Transfer-Encoding

                The  Base64  Content-Transfer-Encoding  is  designed  to
                represent  arbitrary  sequences  of octets in a form that is
                not humanly readable.  The encoding and decoding  algorithms
                are simple, but the encoded data are consistently only about
                33 percent larger than the unencoded data.  This encoding is
                based on the one used in Privacy Enhanced Mail applications,
                as defined in RFC 1113.  The  base64  encoding  is  adapted
                from  RFC  1113, with one change:  base64 eliminates the "*"
```

mechanism for embedded clear text.

A 65-character subset of US-ASCII is used, enabling 6 bits to be represented per printable character. (The extra 65th character, "=", is used to signify a special processing function.)

NOTE: This subset has the important property that it is represented identically in all versions of ISO 646, including US ASCII, and all characters in the subset are also represented identically in all versions of EBCDIC. Other popular encodings, such as the encoding used by the UUENCODE utility and the base85 encoding specified as part of Level 2 PostScript, do not share these properties, and thus do not fulfill the portability requirements a binary transport encoding for mail must meet.

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8-bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first byte, and the eighth bit will be the low-order bit in the first byte, and so on.

Each 6-bit group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string. These characters, identified in Table 1, below, are selected so as to be universally representable, and the set excludes characters with particular significance to SMTP (e.g., ".", "CR", "LF") and to the encapsulation boundaries defined in this document (e.g., "-").

Table 1: The Base64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

The output stream (encoded bytes) must be represented in lines of no more than 76 characters each. All line breaks or other characters not found in Table 1 must be ignored by decoding software. In base64 data, characters other than those in Table 1, line breaks, and other white space

probably indicate a transmission error, about which a warning message or even a message rejection might be appropriate under some circumstances.''

Special processing is performed if fewer than 24 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Output character positions which are not required to represent actual input data are set to the character "=". Since all base64 input is an integral number of octets, only the following cases can arise: (1) the final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output will be an integral multiple of 4 characters with no "=" padding, (2) the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two "=" padding characters, or (3) the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one "=" padding character.

Care must be taken to use the proper octets for line breaks if base64 encoding is applied directly to text material that has not been converted to canonical form. In particular, text line breaks should be converted into CRLF sequences prior to base64 encoding. The important thing to note is that this may be done directly by the encoder rather than in a prior canonicalization step in some implementations.

NOTE: There is no need to worry about quoting apparent encapsulation boundaries within base64-encoded parts of multipart entities because no hyphen characters are used in the base64 encoding.

Base64 was designed for handling messages and files attached to messages. In other words, I have set the limitation of 5mb prior to calling this module. If you call this module directly, this will only handle files less than 2gb.

****

Project: Blowfish Encryption/Decryption

Description: Blowfish is a block cipher that was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms. Like Skipjack it can be used as a convenient and much faster substitute for DES. Since then Blowfish has been extensively analyzed and no significant weaknesses have been found. It is considered to be a strong algorithm and has been implemented in over 130 commercial applications. Bruce Schneier was also one of the designers of the Blowfish algorithm, which is one of the five finalists selected by NIST in the Advanced Encryption Standard (AES) competition. Blowfish was developed as a successor to the Blowfish cipher, which does not meet the design requirements of the AES.

Blowfish is a 64-bit block cipher, meaning that data is encrypted and decrypted in 64-bit chunks. The key length can vary from 32 to 448 bits. The algorithm uses 16 rounds, or iterations of the main algorithm. It has been found that the number of rounds is exponentially proportional to the amount of time required to find a key using a brute-force

attack. So as the number of rounds increases, the security of the algorithm increases exponentially.

Private Encryptor's implementation of Blowfish allows the key to vary from 32 to 448 bits, as per the algorithm specification.

The relative strength of the encryption algorithm is based on key length. Bruce Schneier, creator of the Blowfish encryption algorithm, has calculated that according to what we know of quantum mechanics today, that the entire energy output of the sun is insufficient to break a 197-bit key.

The detailed description of the Blowfish algorithm is contained in the original Blowfish paper, written by Bruce Schneier, which was presented at the First Fast Software Encryption workshop in Cambridge.

****

SPECIAL NOTE: The encryption process has been enhanced.

- Encryption process can be performed multiple times. See property EncryptRounds().

- Password key length is expanded to meet and exceed AES standards. Key lengths are 128 to 448 in 32 bit increments. See property KeyLength().

****

References: Written by Bruce Schneier
http://www.counterpane.com/blowfish.html

Tropical software (Description above)
http://www.tropsoft.com/strongenc/blowfish.htm

An Introduction to Using Keys in Cryptography
By DI Management Services Pty Ltd, Sydney Australia
http://www.di-mgt.com.au/cryptokeys.html

****

Project: GOST 28147-89 Encryption/Decryption

Description: The Government Standard of the U.S.S.R. 28147-89, Cryptographic protection for Data Protection Systems, appears to have played a role in the Soviet Union similar to that played by the U.S. Data Encryption Standard (FIPS 46). When issued, it bore the minimal classification 'For Official Use', but is now said to be widely available in software both in the Former Soviet Union and elsewhere. In apparent contrast to DES's explicit limitation to unclassified information, the introduction to GOST 28147-89 contains the intriguing remark that the cryptographic transformation algorithm "does not place any limitations on the secrecy level of the protected information."

The algorithms are similar in that both operate on 64-bit blocks by successively modifying half of the bits with a function of the other half. Beyond that, the similarity declines and several differences are visible.

- The Soviet System has 32 rounds rather than the 16 of DES.

- Each round is somewhat simpler to a round of DES. In the lMixBits() function, 32 bits of text are added modulo 32 to 32 bits of key, transformed by a block of eight,

4-bit to 4-bit S-boxes and rotated 11 bits to the left.

- In contrast to DES's meagre 56 bits of key, GOST 28147-89 has 256 bits of primary key and 512 bits of secondary key. The secondary key is the block of eight S-boxes, which are specific to individual networks and are not included in the standard (*).

- In place of complex key schedule of DES, the primary key is divided into eight 32-bit words. For the first twenty-four rounds, these are used cyclically in ascended order. For the last eight, they are used in descending order.

The standard is also somewhat broader that FIPS46. It includes output feedback and cipher feedback modes of operation, both limited to 64-bit blocks, and a mode for producing message authentication codes.

The translation is internally colloquial, preferring standard English terms to charmingly quaint cognates of the Russian. The objective is to make it easy for English speaking cryptographers to identify what is novel with minimal effort. In some places, particularly the glossary, with has changed the wording considerably and a few remarks on some of the freer choices, seems in order.

****

SPECIAL NOTE: The encryption process has been enhanced.

- Encryption process can be performed multiple times. See property EncryptRounds().

- Password key length is expanded to meet and exceed AES (Advanced Encryption Standard) standards. Key lengths are 128 to 448 in 32 bit increments and 448 to 1024 in 64 bit increments. See property KeyLength().

- Primary key has been increased in size from eight (8) words to sixteen (16) words. These values are run thru BlockEncrypt() routine so that each value is unique.

- Modified the BlockEncrypt()/BlockDecrypt() routines to be able to access all components of the new primary key. For encryption, first forty-eight (48 = 16 * 3) rounds, primary key is used cyclically in ascending order. For the last sixteen (16), in descending order. Reverse this order for decryption.

- Created multiple sets of S-Box values to be determined by the key length property. See LoadWorkBoxes() routine.

****

References:

Jetico software
http://www.jetico.com/index.htm#/gost.htm

Wikipedia GOST (Block cipher)
http://en.wikipedia.org/wiki/GOST_28147-89

Gost.c
http://vipul.net/gost/software/gost.c

An Introduction to Using Keys in Cryptography
By DI Management Services Pty Ltd, Sydney Australia
http://www.di-mgt.com.au/cryptokeys.html

****

Project: Rijndael Encryption/Decryption
(Pronounced "rine-doll")

Description: The block cipher Rijndael was designed by Joan Daemen and Vincent Rijmen as a candidate for the Advanced Encryption Standard. It was chosen by NIST from a field of 15 candidates. The design of Rijndael was strongly influenced by the design of the block cipher Square, which was also created by Joan Daemen and Vincent Rijmen. The name of the algorithm is a combination of the names of its two creators. The Rijndael web page jokes that the name Rijndael was used "because we were both fed up with people mutilating the pronunciation of the names 'Daemen' and 'Rijmen'". The algorithm can be implemented very efficiently on a wide range of processors and in hardware. Like all AES candidates, Rijndael is very secure and has no known weaknesses.

Rijndael's key length is defined to be either 128, 192, or 256 bits in accordance with the requirements of the AES. Note that unlike Serpent and Twofish, the key size must be one of these values; it is not allowed to be arbitrary. Also unlike other AES candidates, Rijndael has a variable block length of either 128, 192, or 256 bits. All nine combinations of key length and block length are possible, although the official AES block size is 128 bits. Both block length and key length can be extended very easily to multiples of 32 bits. The number of rounds, or iterations of the main algorithm, can vary from 10 to 14 and is dependent on the block size and key length.

Implementation of the AES Rijndael Block Cipher. Inspired by Mike Scott's implementation in C. Permission for free direct or derivative use is granted subject to compliance with any conditions that the originators of the algorithm place on its exploitation.

Rijndael is an iterated block cipher with a variable block length and a variable key length. The block length and the key length can be independently specified to 128, 192 or 256 bits.

Rijndael is a block cipher, designed by Joan Daemen and Vincent Rijmen as a candidate algorithm for the AES .

The cipher has a variable block length and key length. We currently specified how to use keys with a length of 128, 192, or 256 bits to encrypt blocks with al length of 128, 192 or 256 bits (all nine combinations of key length and block length are possible). Both block length and key length can be extended very easily to multiples of 32 bits.

Rijndael can be implemented very efficiently on a wide range of processors and in hardware.

****

SPECIAL NOTE: The encryption process has been enhanced.

- NIST prefixes their encrypted data with orginal data size and other information (approx 16 bytes).
  I append the encrypted data with the original data size (8 bytes) after it has been encrypted by my own algorithm.

- Encryption process can be performed multiple times.
  See property EncryptRounds().

- Password key length and block size has been expanded
  to meet and exceed AES standards. New sizes are 128,
  160, 192, 224, 256 bits. See properties KeyLength()
  and BlockSize().
****

References: The block cipher Rijndael
http://www.ktana.eu/html/theRijndaelPage.htm

NIST home page for the Rijndael block cipher. It has links
to specifications and source code.
http://www.sciencecentral.com/site/497733

Brian Gladman's Home Page
Select cryptography on left panel under technology
http://www.gladman.me.uk/

Tropical software (Portions of description above)
http://www.tropsoft.com/strongenc/rijndael.htm

Rijndael converted to Visual Basic 6.0
John Korejwa <korejwa@tiac.net>
09-Jul-2006

http://www.Planet-Source-Code.com/vb/scripts/ShowCode.asp?txtCodeId=65906&lngWId=1

An Introduction to Using Keys in Cryptography
By DI Management Services Pty Ltd, Sydney Australia
http://www.di-mgt.com.au/cryptokeys.html

****

Project: Serpent Encryption/decryption

Description: Serpent was designed by Ross Anderson, Eli Biham and
Lars Knudsen as a candidate for the Advanced Encryption
Standard. It was been selected as one of the five
finalists in the AES competition. Serpent is faster
than DES and more secure than Triple DES. It provides
users with a very high level of assurance that no
shortcut attack will be found. To achieve this, the
algorithm 's designers limited themselves to well
understood cryptography mechanisms, so that they could
rely on the wide experience and proven techniques of
block cipher cryptanalysis. The algorithm uses twice as
many rounds as are necessary to block all currently
known shortcut attacks. This means that Serpent should
be safe against as yet unknown attacks that may be
capable of breaking the standard 16 rounds used in many
types of encryption today. However, the fact that
Serpent uses so many rounds means that it is the
slowest of the five AES finalists. But this shouldn't
be an issue because it still outperforms Triple DES.
the algorithm 's designers maintain that Serpent has a
service life of at least a century."

Serpent is a 128-bit block cipher, meaning that data is
encrypted and decrypted in 128-bit chunks. The key
length can vary, but for the purposes of the AES it is
defined to be either 128, 192, or 256 bits. This block
size and variable key length is standard among all AES
candidates and was one of the major design requirements

specified by NIST. The Serpent algorithm uses 32 rounds, or iterations of the main algorithm."

In this module, Serpent uses a 256 bit key (256 = 8 * 32 bytes). If a shorter key is provided by the user, the Serpent algorithm itself pads the key to make it 256 bits long.

Like DES, Serpent includes an initial and final permutation of no cryptographic significance; these permutations are used to optimize the data before encryption. The detailed description of the actual algorithm is contained in the official Serpent paper submitted for the AES by the algorithm's designers.

****

SPECIAL NOTE: The encryption process has been enhanced.

- Encryption process can be performed multiple times. See property EncryptRounds().

- Password key length is expanded to meet and exceed AES standards. Key lengths are 128 to 448 in 32 bit increments and 448 to 1024 in 64 bit increments. See property KeyLength().

****

References: Serpent Encryption Algorithm
http://www.cl.cam.ac.uk/~rja14/serpent.html

The algorithm specifications.
http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf

Full submission package for AES
http://www.cl.cam.ac.uk/~rja14/Papers/serpent.tar.gz

Most of the description above was obtained from
http://www.tropsoft.com/strongenc/serpent.htm

An Introduction to Using Keys in Cryptography
By DI Management Services Pty Ltd, Sydney Australia
http://www.di-mgt.com.au/cryptokeys.html

****

Project: Skipjack Encryption/Decryption

Description: Skipjack is a recent algorithm that was developed in 1987 and put into service in 1993. It is a formerly secret NSA (National Security Agency)encryption algorithm that was declassified on June 23, 1998. Skipjack is a representative of a family of encryption algorithms developed in 1980 as part of the NSA suite of "Type I" algorithms, which are suitable for protecting all levels of classified data. Type I algorithms are typically extremely secure and are usually classified as secret. Skipjack was used to encrypt sensitive, but not classified, government data. It was implemented in two government encryption devices: the Clipper chip and Fortezza PC card. These devices have many uses and are widely employed by agencies such as the FBI and NSA. They provide a high level of security for sensitive communications while enabling the interception of telecommunications by law enforcement officials for such things as criminal investigations. For example, Clipper

chips can be used to provide secure telephone transmissions and Fortezza cards can be used to encrypt such things as e-mail and network traffic. The key characteristic of both devices is that they were designed with "back doors" that allow government agents to monitor encrypted transmissions given the proper authority. This feature is covered under the Escrowed Encryption Standard, of which Skipjack is a part. It is implemented via a mechanism called a LEAF (Law Enforcement Access Field). It is important to note that Skipjack itself has nothing to do with this "back door" functionality! Skipjack is entirely separate from the LEAF in the Clipper and Fortezza products and is not affected in any way by its presence.

Skipjack has been extensively cryptanalyzed, and has no weaknesses. There are no known shortcut attacks that can break Skipjack. However, the small key size makes this algorithm inferior to the newer candidate algorithms for the AES (Advanced Encryption Standard) competition being held by NIST (National Institute of Standards and Technology). The NIST is a federal technology agency that develops and promotes measurement, standards, and technology. Despite this shortcoming, Skipjack still provides very strong security and it should be many years before the algorithm is broken by a brute force attack. Note that it was declassified in order to provide a software implementation of Fortezza enabled applications. The NSA does not intend for the algorithm to be a candidate for the AES. Like Triple DES, Skipjack is an interim solution to be used until the final AES is completed and widely implemented. It offers a safe alternative to DES without having to rely on the AES.

In 1993 an Interim Report was released that gave a thorough analysis of Skipjack and discussed issues relating to Skipjack's integration in the Clipper and Fortezza hardware. Although this report is many years old, it provides some interesting information and a good perspective on what it would take to break Skipjack with a brute force attack. While reading the document, keep in mind that the algorithm was still a closely guarded secret at the time the report was written. The Final Report mentioned in this document was never written, so this document is the closest thing there is to an official analysis of the Skipjack algorithm.

Skipjack encrypts and decrypts data in 64-bit blocks, normally using an 80-bit key. It takes a 64-bit block of plaintext as input and outputs a 64-bit block of ciphertext. Skipjack has 32 rounds, meaning the main algorithm is repeated 32 times to produce the ciphertext. It has been found that the number of rounds is exponentially proportional to the amount of time required to find a key using a brute-force attack. So as the number of rounds increases, the security of the algorithm increases exponentially.

The detailed description of the actual algorithm is contained in the official Skipjack specification provided by the NSA after the algorithm was declassified. Another algorithm called KEA was declassified at the same time As Skipjack, so the specification contains information about both Skipjack and KEA. The first half of the document concerns Skipjack and the second half focuses on KEA (Key

Exchange Algorithm).

****

SPECIAL NOTE: The encryption process has been enhanced.

- Encryption process can be performed multiple times. See property EncryptRounds().

- Password key length is expanded to meet and exceed AES standards. Key lengths are 128 to 448 in 32 bit increments and 448 to 1024 in 64 bit increments. See property KeyLength().

- F-table loading is more dynamic because it is now based on key length and not a single table. See LoadTable() routine.

****

Note: I have found several C versions of Skipjack on the web. My interpretation is based on the version by Paulo Barreto with some modifications by myself because the C language can manipulate numeric values that are not easily done within the realm of Visual Basic 6.0.

Originally written by Panu Rissanen <bande@lut.fi> 1998.06.24
Optimized by Mark Tillotson <markt@chaos.org.uk> 1998.06.25
Optimized by Paulo Barreto <pbarreto@nw.com.br> 1998.06.30

http://www.larc.usp.br/~pbarreto/
http://www.larc.usp.br/~pbarreto/skipjack32.zip

References: Skipjack documentation above
http://csrc.nist.gov/encryption/skipjack/skipjack.pdf

Tropical software (Portions of description above)
http://www.tropsoft.com/strongenc/skipjack.htm

Observations on the SkipJack Encryption Algorithm
http://www.cs.technion.ac.il/~biham/Reports/SkipJack/

An Introduction to Using Keys in Cryptography
By DI Management Services Pty Ltd, Sydney Australia
http://www.di-mgt.com.au/cryptokeys.html

****

Project: Twofish Encryption/Decryption

Description: Twofish is a block cipher by Counterpane Labs. It was one of the five Advanced Encryption Standard (AES) finalists. The Twofish cipher has not been patented and the reference implementation has been placed in the public domain. As a result, the Twofish algorithm is free for anyone to use without any restrictions whatsoever.

Twofish is a 128-bit block cipher that accepts a variable-length key up to 256 bits. The Twofish block cipher is Counterpane Labs' candidate for the new Advanced Encryption Standard. It was one of the five finalists chosen by NIST from a field of 15 candidates as explained above. Twofish is designed to be highly secure and highly flexible. It is well suited for large microprocessors, 8-bit smart card microprocessors, and dedicated hardware. Counterpane Labs has spent over one thousand hours cryptanalyzing Twofish, and has found no attacks that can break the full 16 round version of the algorithm. The algorithm is very

secure when the full 16 rounds are used. It was also the fastest
AES candidate, and one of the most compact. Its conservative
design allows the ability to trade off key setup time for
encryption speed, as well as sacrificing smaller memory
requirements to obtain greater encryption speed. Algorithm
implementers are given a lot of flexibility to play around with
to produce a version of Twofish that is just right for the
job at hand.

Twofish was designed by Bruce Schneier, John Kelsey,
Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson.
Bruce Schneier also designed the Blowfish algorithm, which
remains unbroken after eight years of cryptanalysis and has
been implemented in over 130 commercial applications.

****
SPECIAL NOTE: The encryption process has been enhanced.

- Encryption process can be performed multiple times.
  See property EncryptRounds().

- Password key length is expanded to meet and exceed AES
  standards. Key lengths are 128 to 448 in 32 bit
  increments. See property KeyLength().
****

References: Twofish was created and analyzed by:
Bruce Schneier, John Kelsey, Doug Whiting, David Wagner,
Chris Hall, Niels Ferguson
http://www.schneier.com/Twofish.html

Tropical software (Portions of description above)
http://www.tropsoft.com/strongenc/Twofish.htm

Visual Basic translation by Fredrik Qvarfort

http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=12023&lngWId=1
12-Oct-2000

An Introduction to Using Keys in Cryptography
By DI Management Services Pty Ltd, Sydney Australia
http://www.di-mgt.com.au/cryptokeys.html

****

****
** Overview Hash modules
****

All hash algorithms have the capability performing multiple looping for
a single hash output. See property HashRounds() within each module.

****

Module: clsAPI_Hash (CryptoAPI)

Special note: SHA-224, SHA-512/224 and SHA-512/256 have not yet been
implemented into the Microsoft crypto suite of hashes.

Description: This is a class which accesses Microsoft's CryptoAPI hash
algorithms. These include:

MD2 MD4 MD5 SHA-1 SHA-256 SHA-384 SHA-512

The Secure Hash Algorithm (SHA) is required for use with the Digital Signature Algorithm (DSA) as specified in the Digital Signature Standard (DSS) and whenever a secure hash algorithm is required for federal applications. For a message of length < 2^64 bits, this algorithm produces a condensed representation of the message called a message digest. The message digest is used during generation of a signature for the message. This also used to compute a message digest for the received version of the message during the process of verifying the signature. Any change to the message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify.

The MD5 algorithm is an extension of the MD4 message-digest algorithm 1,2. MD5 is slightly slower than MD4, but is more "conservative" in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations. The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard.

These algorithms have been tested to be accurate in accordance with FIPS 180-2 AND FIPS 180-3 publication.

REFERENCE: The Cryptography API, or How to Keep a Secret
http://msdn.microsoft.com/en-us/library/ms867086.aspx

CryptoAPI Cryptographic Service Providers
http://msdn.microsoft.com/en-us/library/bb931357(VS.85).aspx

SHA-2 support on MS Windows
Excert, "Regarding SHA-224 support, SHA-224 offers less security than SHA-256 but takes the same amount of resources. Also SHA-224 is not generally used by protocols and applications. The NSA's Suite B standards also do not include it. We have no plans to add it on future versions of our CSPs."

http://blogs.msdn.com/b/alejacma/archive/2009/01/23/sha-2-support-on-windows-xp.aspx

****

Module: Tiger3 Hash Algorithm

This module is my experimental version for Tiger-128 thru Tiger-512 bit output. I call it Tiger3. I have not had any problems with it thus far. If you should encounter any problems, please email me at:

Kenneth Ives kenaso@tx.rr.com

DO NOT CONTACT Ross Anderson, http://www.cl.cam.ac.uk/users/rja14/
Eli Biham, http://www.cs.technion.ac.il/~biham

because they did not write this module nor are they responsible in any manner as to its content.

Description: Tiger2 is a fast New cHash function, designed to be very fast

on modern computers, and in particular on the state-of-the-art 64-bit computers (like DEC-Alpha), while it is still not slower than other suggested hash functions on 32-bit machines.

Tiger hash has no usage restrictions nor patents. It can be used freely, with the reference implementation, with other implementations or with a modification to the reference implementation (as long as it still implements Tiger2). The authors only ask that you to let them know about your implementation and to cite the origin of Tiger and of the reference implementation.
http://www.cs.technion.ac.il/~biham/Reports/Tiger/

My opinion: Tiger2 is a very strong hash. Unfortunately, the 32-bit version seems to be just an off shoot and is not the primary concern of the authors. That is understandable because everyone appears to be headed for 64-bit hash scenarios. The test vectors, I believe, were created prior to the new MD5 padding of Tiger2 and may be obsolete. The time stamp on the 32-bit version of the Tiger code is dated 30-Mar-1996. When I contacted one of the authors, the response was less than enthusiastic concerning the 32-bit version of Tiger2.

Special Note: I decided to make changes to the initial work data array [malngHash()]. I believe that this is a cleaner and more secure method as to the calculation of the Tiger hashes. See LoadWorkArrays() routine for more details.

Reference: Tiger2 - A Fast New Hash function
http://www.cs.technion.ac.il/~biham/Reports/Tiger/

Original authors of the Tiger hash:
Ross Anderson, http://www.cl.cam.ac.uk/users/rja14/
Eli Biham, http://www.cs.technion.ac.il/~biham

Tiger2 32-bit C source code at:
http://www.cs.technion.ac.il/~biham/Reports/Tiger/tiger-src32.zip

Tiger2 64-bit Source code dtd 8-Feb-1996
http://www.cs.technion.ac.il/~biham/Reports/Tiger/tiger-src.zip

****

Module: Whirlpool Hash

Whirlpool versions for -224, -256, -384 bit output are my experiment. Any problems with these outputs, please email me at:

Kenneth Ives kenaso@tx.rr.com

DO NOT CONTACT Vincent Rijmen
Paulo S. L. M. Barreto

because they did not write this module nor are they responsible in any manner as to its content.

Description: The WHIRLPOOL Hashing Function
Designed by Vincent Rijmen and Paulo S. L. M. Barreto

Whirlpool is a cryptographic hash function adopted by the International Standards Organization (ISO) and International Electrotechnical Commission (IEC) as part of the joint ISO/IEC 10118-3 international standard. It takes an arbitrary block of data and returns a 512 bit digest that can be used as a digital fingerprint for

message authentication. Compliance with the standard may be verified at:
http://hash.online-convert.com/whirlpool-generator

Whirlpool is a hash designed after the Square block cipher. Whirlpool is a Miyaguchi-Preneel construction based on a substantially modified Advanced Encryption Standard (AES). It takes a message of any length less than 2256 bits and returns a 512-bit message digest.

The authors have declared that "WHIRLPOOL is not (and will never be) patented and may be used free of charge for any purpose. The reference implementations are in the public domain."

The algorithm is named after the Whirlpool Galaxy in Canes Venatici.

Using the reference C implementation on a 1 GHz Pentium III platform, we observe that Whirlpool operates at about 73 cycles per hashed byte.

The compression function runs at about 56 cycles per hashed byte. Many factors explain the observed performance. First, a 32-bit processor was used to test a native 64-bit implementation; better results are expected by merely running the speed measurement on an Alpha or Itanium processor. Second, it seems that the pipe parallelism capabilities of the Pentium were not fully used; this may reflect a non-optimising implementation of 64-bit arithmetic support by the C compiler, and might be overcome by an assembler implementation. Third, the tables employed in the reference implementation are quite large, and the built-in processor cache might not be enough to hold them, the data being hashed, and the hashing code at once, thus degrading processing speed.

Whirlpool is much more scalable than most modern hashing functions. Even though is not specifically oriented toward any platform, it is rather efficient on many of them, its structure favouring extensively parallel execution of the component mappings. At the same time, it does not require excessive storage space (either for code or for tables), and can therefore be efficiently implemented in quite constrained environments like smart cards, although it can benefit from larger cache memory available on modern processors to achieve higher performance. It does not use expensive or unusual instructions that must be built in the processor. The mathematical simplicity of the primitive resulting from the design strategy tends to make analysis easier. And finally, it has a very long hash length; this not only provides increased protection against birthday attacks, but also offers a larger internal state for entropy containment, as is needed for certain classes of pseudo-random number generators.

Reference: The WHIRLPOOL Hash Function (Home page)
http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html

Whirlpool (cryptography)
http://en.wikipedia.org/wiki/Whirlpool_(cryptography)

```
            Whirlpool Hashing Function in Visual Basic 6.0
            Author:  Korejwa2   12/31/2010

http://www.Planet-Source-Code.com/vb/scripts/ShowCode.asp?txtCodeId=73638&lngWId=1
```

****
** Overview Miscellaneous modules
****

Random data is generated using a PRNG (Pseudo Random Number Generator). This is a cryptographically random number generator that comes with Microsoft Internet Explorer called CryptoAPI. CryptGenRandom gets its randomness, also known as entropy, from many sources in Windows, including the following:

- The current process ID (GetCurrentProcessID).
- The current thread ID (GetCurrentThreadID).
- The number of milliseconds since the last boot (GetTickCount).
- The current time (GetLocalTime).
- Various high-precision performance counters (QueryPerformanceCounter).
- A Message Digest-4 (MD4) hash of the user's environment block, which includes username, computer name, and search path. MD4 is a hashing algorithm that creates a 128-bit message digest (16 bytes) from input data to verify data integrity.
- High-precision internal CPU counters, such as RDTSC, RDMSR, RDPMC.
- Low-level system information, such as idle time, kernel time, interrupt times, commit limit, page read count, cache read count, nonpaged pool allocations, alignment fixup count, operating system lookaside information.
- [Optional] User defined data as extra seed data. I created a routine named GetExtraSeed() to generate a unique 40 byte hex data string as my optional data.

Such information is added to a buffer, which is hashed using MD4 and used as the key to modify a buffer, using RC4, using RC4. (Refer to the CryptGenRandom() documentation in
the Platform SDK) The result is a cryptographic random the user-provided buffer.) The result is a cryptographic random number generator.

Note from Mark Hutchinson's presentation about Microsoft's VB random number generator.

References: Randomize Statement Doesn't Re-initialize Rnd Function
http://support.microsoft.com/default.aspx?scid=kb;en-us;120587

"To re-initialize the random-number generator, use the Rnd function with a
value of -1 to re-initialize the Rnd function, and then use the Randomize
statement with the value you want to use as the seed value for the Rnd function."

VBA 's Pseudo Random Number Generator
http://www.noesis.net.au/prng.php

INFO: How Visual Basic Generates Pseudo-Random Numbers for the RND Function
http://support.microsoft.com/kb/231847/en-us

RND and RANDOMIZE Alternatives for Generating Random Numbers
http://support.microsoft.com/kb/28150/EN-US/

****

Cyclic Redundancy Check 32-bit (CRC32)

The CRC is a very powerful but easily implemented technique to obtain data reliability. The CRC technique is used to protect blocks of data called Frames. Using this technique, the transmitter appends an extra n- bit sequence to every frame called Frame Check Sequence (FCS). The FCS holds redundant information about the frame that helps the transmitter detect errors in the frame. The CRC is one of the most used techniques for error detection in data communications. The technique gained its popularity

because it combines three advantages:

- Extreme error detection capabilities
- Little overhead
- Ease of implementation

A CRC checksum must be a full 8 hex characters otherwise leading zeroes
are not required.

****

Preamble

This License governs Your use of the Work. This License is intended to allow developers to use the Source Code and Executable Files provided as part of the Work in any application in any form.

The main points subject to the terms of the License are:

    Source Code and Executable Files can be used in commercial applications;
    Source Code and Executable Files can be redistributed; and
    Source Code can be modified to create derivative works.

No claim of suitability, guarantee, or any warranty whatsoever is provided. The software is provided "as-is".

This License is entered between You, the individual or other entity reading or otherwise making use of the Work licensed pursuant to this License and the individual or other entity which offers the Work under the terms of this License ("Author").

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CODE PROJECT OPEN LICENSE ("LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HEREIN, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE AUTHOR GRANTS YOU THE RIGHTS CONTAINED HEREIN IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO ACCEPT AND BE BOUND BY THE TERMS OF THIS LICENSE, YOU CANNOT MAKE ANY USE OF THE WORK.

Definitions.

"Articles" means, collectively, all articles written by Author which describes how the Source Code and Executable Files for the Work may be used by a user.

"Author" means the individual or entity that offers the Work under the terms of this License.

"Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works.

"Executable Files" refer to the executables, binary files, configuration and any required data files included in the Work.

"Publisher" means the provider of the website, magazine, CD-ROM, DVD or other medium from or by which the Work is obtained by You.

"Source Code" refers to the collection of source code and configuration files used to create the Executable Files.

"Standard Version" refers to such a Work if it has not been

modified, or has been modified in accordance with the consent of the Author, such consent being in the full discretion of the Author.

"Work" refers to the collection of files distributed by the Publisher, including the Source Code, Executable Files, binaries, data files, documentation, whitepapers and the Articles.

"You" is you, an individual or entity wishing to use the Work and exercise your rights under this License.

Fair Use/Fair Use Rights. Nothing in this License is intended to reduce, limit, or restrict any rights arising from fair use, fair dealing, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

License Grant. Subject to the terms and conditions of this License, the Author hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

You may use the standard version of the Source Code or Executable Files in Your own applications.

You may apply bug fixes, portability fixes and other modifications obtained from the Public Domain or from the Author. A Work modified in such a way shall still be considered the standard version and will be subject to this License.

You may otherwise modify Your copy of this Work (excluding the Articles) in any way to create a Derivative Work, provided that You insert a prominent notice in each changed file stating how, when and where You changed that file.

You may distribute the standard version of the Executable Files and Source Code or Derivative Work in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution.

The Articles discussing the Work published in any form by the author may not be distributed or republished without the Author's consent. The author retains copyright to any such Articles. You may use the Executable Files and Source Code pursuant to this License but you may not repost or republish or otherwise distribute or make available the Articles, without the prior written consent of the Author.

Any subroutines or modules supplied by You and linked into the Source Code or Executable Files this Work shall not be considered part of this Work and will not be subject to the terms of this License.

Patent License. Subject to the terms and conditions of this License, each Author hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, import, and otherwise transfer the Work.

Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

You agree not to remove any of the original copyright, patent, trademark, and attribution notices and associated disclaimers

that may appear in the Source Code or Executable Files.

You agree not to advertise or in any way imply that this Work is a product of Your own.

The name of the Author may not be used to endorse or promote products derived from the Work without the prior written consent of the Author.

You agree not to sell, lease, or rent any part of the Work.

You may distribute the Executable Files and Source Code only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy of the Executable Files or Source Code You distribute and ensure that anyone receiving such Executable Files and Source Code agrees that the terms of this License apply to such Executable Files and/or Source Code. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute the Executable Files or Source Code with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License.

You agree not to use the Work for illegal, immoral or improper purposes, or on pages containing illegal, immoral or improper material. The Work is subject to applicable export laws. You agree to comply with all such laws and regulations that may apply to the Work after Your receipt of the Work.

Representations, Warranties and Disclaimer. THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABLE QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

Indemnity. You agree to defend, indemnify and hold harmless the Author and the Publisher from and against any claims, suits, losses, damages, liabilities, costs, and expenses (including reasonable legal or attorneys’ fees) resulting from or relating to any use of the Work by You.

Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL THE AUTHOR OR THE PUBLISHER BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK OR OTHERWISE, EVEN IF THE AUTHOR OR THE PUBLISHER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Termination.

This License and the rights granted hereunder will terminate automatically upon any breach by You of any term of this License. Individualss or entities who have received Derivative Works from

You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 6, 7, 8, 9, 10 and 11 will survive any termination of this License.

If You bring a copyright, trademark, patent or any other infringement claim against any contributor over infringements You claim are made by the Work, your License from such contributor to the Work ends automatically.

Subject to the above terms and conditions, this License is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, the Author reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

Publisher. The parties hereby confirm that the Publisher shall not, under any circumstances, be responsible for and shall not have any liability in respect of the subject matter of this License. The Publisher makes no warranty whatsoever in connection with the Work and shall not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. The Publisher reserves the right to cease making the Work available to You at any time without notice

Miscellaneous.

This License shall be governed by the laws of the location of the head office of the Author or if the Author is an individual, the laws of location of the principal place of residence of the Author.

If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this License, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

This License constitutes the entire agreement between the parties with respect to the Work licensed herein. There are no understandings, agreements or representations with respect to the Work not specified herein. The Author shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Author and You.