mediaCrypt®

Internet, Multimedia,
Wireless Communications and
CRM Systems

Highly Secure Encryption

Cyber Security

IDcypher®

Advanced
Blockchain Technology

v2018.3.831

(for .NET 4.x)
released on August 31, 2018

Internet, Multimedia, Wireless Communications and CRM systems

Highly secure encryption systems, Military Grade Encryption

Mediacrypt.IDcypher

(For .NET 4.x) released on August 31, 2018

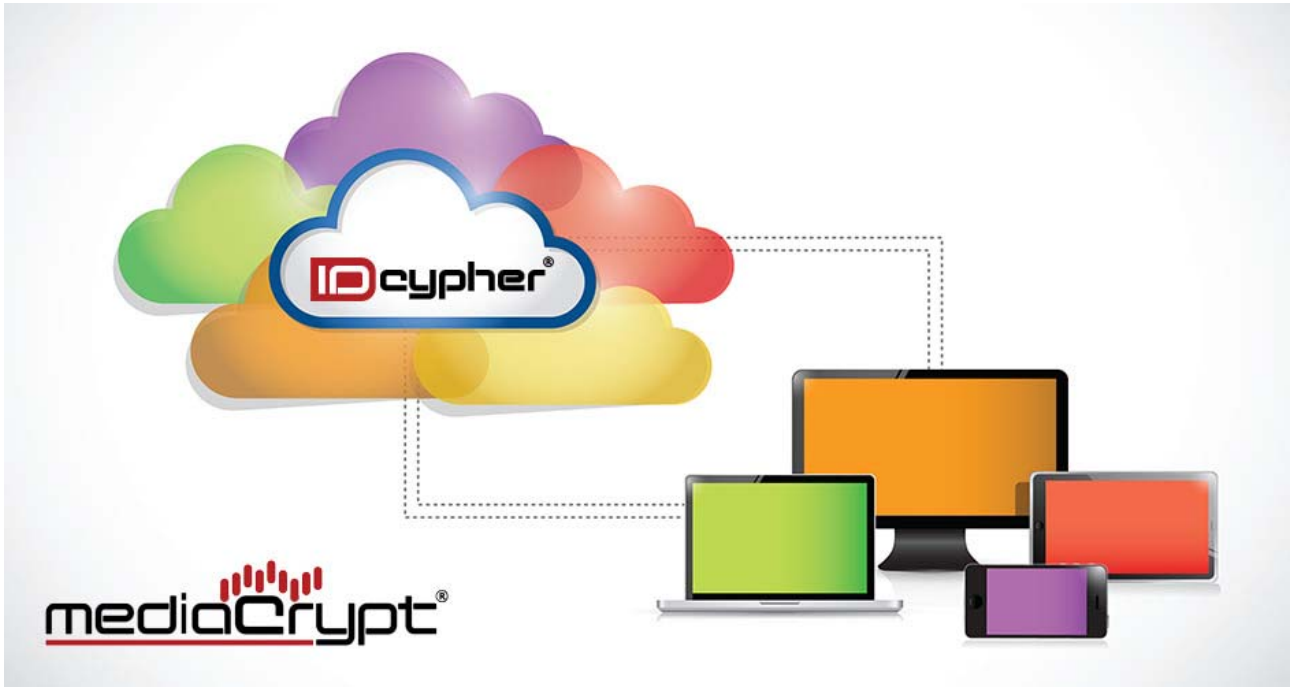
Quick Reference

Revision: **001**
Date: **August 31, 2018**
Reference: **PB_SW_IDC_2018.3.831_QRF_001**

Document Index

The Document Index shows an overview of this document. It lets you navigate rapidly.

#1 Introduction



'In business, having the right information at the right time can make the difference between profit and loss, success and failure.'

IDcypher is a security-related component library, which lets you quickly and easily incorporate strong encryption into your application.

The three pillars of **Information Security**:

Confidentiality: protecting information from unauthorised disclosure;

Integrity: protecting information from unauthorised modifications, and ensure that information is accurate and complete;

Availability: ensuring information is available when needed.

Developers are able to create spectacularly powerful applications, which transform everyday life for individuals, but when a company's assets or reputation are on the line the stakes become much higher. In the information age and in the interdependent digital economy that is driving globalization and progress today, protecting a company's sensitive files in any way possible is the key to survival. There are competitors who are willing to try almost any means to get at your data, and one of the most damaging breaches that a company can suffer is that of intellectual property that has been painstakingly developed and perfected being appropriated or misused at the hands of a competitor.

Intellectual property is under siege through the actions of hackers who are able to access systems remotely and with relative immunity to repercussion. The victims of this crime often do not get redress. Financial systems are particularly sensitive to being hacked because of the large amounts of very valuable data that they process. For a corporation, the compromising of intellectual property to a significant enough degree could carry a price tag in the millions. Developers must arm themselves against the various forces that are working against them. Cryptographic methods have been long utilized to protect business. But technology can also be used against an enterprise. IDcypher is designed to combat this threat.

Mediacrypt.IDcypher is a newly developed security library, which is now available to the general developer community. Containing an array of powerful tools, hash functions, and ciphers, IDcypher is designed to bring a complete package of encryption and security aids easily to hand. Data compression and encryption are accomplished with platform-independent code that will work in 32 bit and 64 bit environments. Advanced features such as Encrypted Storage provide reliable protection for structured and file-based data.

Standard .NET classes are used as a template for library components. This serves the dual purposes not only of adding encryption features to newly developed programs, but additionally it makes the refactoring of encryption to existing programs a far more streamlined process. Basic and multi-threaded ciphers can be loaded into the IDcypher library side by side, and are automatically accessed by the system, which is using the library. The ability of IDcypher to load compatible components transparently makes easy customization to the target environment possible simply by copying the required component set. IDcypher can then adapt on the fly to the target operating system, number of cores, hardware configuration, and so on.

IDcypher includes a unique Key Manager component, which will transparently handle user-supplied passwords and binary keys. This will save an enormous amount of time and headaches for developers otherwise forced to write code for the management of passwords and keys.

Prominent in the IDcypher toolkit is the file shredder, which securely removes data after it has been encrypted. The shredder overwrites data a given number of times, making data recovery less feasible with each pass. After the data is completely erased, the shredder deletes the file, removing every trace of its directory record. The file data is scrambled to such an extent that the original file cannot be reconstituted by any means.

With all these capabilities encapsulated in one package, IDcypher is the preeminent tool for developers today to secure important intellectual property from those who would misuse it. Applications must be secured to ensure that your business is secure. IDcypher from mediacrypt.com is your secret weapon against hackers.

IDcypher does not contain unmanaged or unsafe code; all the components are platform-independent and can be run in both 32-bit and 64-bit environments (with exception of a few optional platform-optimized components).

Mediacrypt.IDcypher Library (High Performance Multi-Threaded, Military Grade Encryption, Patent pending Key Management).

IDcypher and the **IDcypher Logo** are registered trademarks of Trust 47 Fund (Registration Number: **302016000012990**).

#2 Features

Feature	Description
Ciphers	Components that encrypt and decrypt data using a secret key.
Compressors	Components that perform a lossless compression of a data stream.
Encoders	Convert binary data to a text string and vise versa.
Hash Functions	Compute a hash or a message digest of data useful for authencity/error checking.
Random Number Generators	Components for producing cryptographically strong sequences of random numbers.
Shredders	Securely destroy sensitive information. These components are required for file encryption.
Key Manager	Helps manage user-supplied passwords and keys.
Encrypted Storage	Components implementing high-level encrypted storage.
Documentation	Industry standard formatted help is included.
Royalty Free Distribution	Include with any .NET project royalty free.
Lifetime Support	Lifetime E-mail technical support is included.

#3 First Steps

In order to use IDcypher in your application you need only two simple steps:

1. Add a reference to IDcypher.

Right-click your project and select 'Add Reference...'. Select the Browse tab, browse to the directory that contains IDcypher binaries (by default it is 'Mediacrypt.IDcypher.Assemblies' folder), and add a reference to IDcypherFull.dll.

2. Import IDcypher types.

Example

Visual Basic

```
Imports Mediacrypt.IDcypher
```

C#

```
using Mediacrypt.IDcypher;
```

That's all! Now you can create and use IDcypher components like any other type:

Example

Visual Basic

```
Dim aes As CipherAes = New CipherAes
```

C#

```
CipherAes aes = new CipherAes();
```

Remember, however, that IDcypher is an interface-based library, and that some functions are available only as interface members to avoid name conflicts (this is true only for the encryption interfaces of block ciphers; other components don't require casting to interfaces).

Example

Visual Basic

```
Dim aes As CipherAes = New CipherAes

'Error: 'EncryptString' is not a member of 'Mediacrypt.IDcypher.CipherAes'.
Dim encrStr As Byte() = aes.EncryptString("Some string to encrypt.")

'Correct
Dim encrStr As Byte() = CType(aes, IMemoryBlockCipher).EncryptString("Some string to
encrypt.")
```

C#

```
CipherAes aes = new CipherAes();

// Error: 'Mediacrypt.IDcypher.CipherAes' does not contain a definition for 'EncryptString'
byte[] encrStr = aes.EncryptString("Some string to encrypt.");

// Correct
byte[] encrStr = ((IMemoryBlockCipher)aes).EncryptString("Some string to encrypt.");
```

Your code might look simpler if you instantiate a component as the required interface:

Example

Visual Basic

```
Dim aes As IMemoryBlockCipher = New CipherAes
Dim encrStr As Byte() = aes.EncryptString("Some string to encrypt.")
```

C#

```
IMemoryBlockCipher aes = new CipherAes();
byte[] encrStr = aes.EncryptString("Some string to encrypt.");
```

Now you know all you need to start using IDcypher. See the Library Components topics for a detailed discussion of IDcypher components.

#4 Library Components

This topic contains the complete list of IDcypher components. The table includes the following info:

Guid

The component Guid is required for dynamic component creation using the function CreateComponent. For example, in order to create dynamically a multii-threaded AES cipher, use the following call:

Example

Visual Basic

```
Dim aes As Object = CsComponents.CreateComponent(Crypto.CipherAesMt)
```

C#

```
object aes = CsComponents.CreateComponent(Crypto.CipherAesMt);
```

Value

This column contains the actual component Guid value. It is shown for the reference only.

Class Name

The name of the component class. It is required for static component creation, using the new operator:

Example

Visual Basic

```
Dim aes As Object = New CipherAesMt()
```

C#

```
object aes = new CipherAesMt();
```

File

The component's assembly name. In order to use the component this assembly must be present in the application directory. If you create the component as a class using the new operator, add this assembly to the References list.

NOTE: The alternate method is to reference IDcypherFull.dll, which contains all the components.

Ciphers

Guid	Class Name	File	Description
CipherAes **CipherAesMt**	CipherAes CipherAesMt	CipherAes.dll CipherAesMt.dll	Rijndael by Joan Daemen and Vincent Rijmen. This is a superset of the AES encryption algorithm, offering additional modes with 196-bit and 256-bit block sizes.
CipherBlowfish **CipherBlowfishMt**	CipherBlowfish CipherBlowfishMt	CipherBlowfish.dll CipherBlowfishMt.dll	Blowfish by Bruce Schneier.
CipherDes **CipherDesMt**	CipherDes CipherDesMt	CipherDes.dll CipherDesMt.dll	DES (Data Encryption Standard).
CipherSerpent **CipherSerpentMt**	CipherSerpent CipherSerpentMt	CipherSerpent.dll CipherSerpentMt.dll	An AES candidate cipher by Ross Anderson, Eli Biham, and Lars Knudsen.
CipherTripleDes **CipherTripleDesMt**	CipherTripleDes CipherTripleDesMt	CipherTripleDes.dll CipherTripleDesMt.dll	Triple-DES
CipherTwofish **CipherTwofishMt**	CipherTwofish CipherTwofishMt	CipherTwofish.dll CipherTwofishMt.dll	An AES candidate cipher from Bruce Schneier.

Guid	Value
CipherAes **CipherAesMt**	{11056249-400A-4461-BD5E-FE06113A1001} {11056249-400A-4461-BD5E-FE0611BA1001}
CipherBlowfish **CipherBlowfishMt**	{11056249-400A-4461-BD5E-FE06113A1002} {11056249-400A-4461-BD5E-FE0611BA1002}
CipherDes **CipherDesMt**	{11056249-400A-4461-BD5E-FE06113A1003} {11056249-400A-4461-BD5E-FE0611BA1003}
CipherSerpent **CipherSerpentMt**	{11056249-400A-4461-BD5E-FE06113A1006} {11056249-400A-4461-BD5E-FE0611BA1006}
CipherTripleDes **CipherTripleDesMt**	{11056249-400A-4461-BD5E-FE06113A1004} {11056249-400A-4461-BD5E-FE0611BA1004}
CipherTwofish **CipherTwofishMt**	{11056249-400A-4461-BD5E-FE06113A1007} {11056249-400A-4461-BD5E-FE0611BA1007}

Hash Functions

Guid	Class Name	File	Description
HashHaval **HashHavalMt**	HashHaval HashHavalMt	HashHaval.dll HashHavalMt.dll	HAVAL hash function by Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry.
HashMD5 **HashMD5Mt**	HashMD5 HashMD5Mt	HashMD5.dll HashMD5Mt.dll	MD5 message digest function by Ronald Rivest, RFC1321.
HashRipemd **HashRipemdMt**	HashRipemd HashRipemdMt	HashRipemd.dll HashRipemdMt.dll	RIPEMD-128 and RIPEMD-160 hash functions by Antoon Bosselaers, Katholieke Universiteit Leuven.
HashSha1 **HashSha1Mt**	HashSha1 HashSha1Mt	HashSha1.dll HashSha1Mt.dll	Secure Hash Algorithm as defined in FIPS 180-2.
HashSha256 **HashSha256Mt**	HashSha256 HashSha256Mt	HashSha256.dll HashSha256Mt.dll	Secure Hash Algorithm as defined in FIPS 180-2.
HashSha384 **HashSha384Mt**	HashSha384 HashSha384Mt	HashSha384.dll HashSha384Mt.dll	Secure Hash Algorithm as defined in FIPS 180-2.
HashSha512 **HashSha512Mt**	HashSha512 HashSha512Mt	HashSha512.dll HashSha512Mt.dll	Secure Hash Algorithm as defined in FIPS 180-2.

Guid	Value
HashHaval **HashHavalMt**	{11056249-400A-4461-BD5E-FE06113A1021} {11056249-400A-4461-BD5E-FE0611BA1021}
HashMD5 **HashMD5Mt**	{11056249-400A-4461-BD5E-FE06113A1022} {11056249-400A-4461-BD5E-FE0611BA1022}
HashRipemd **HashRipemdMt**	{11056249-400A-4461-BD5E-FE06113A102D} {11056249-400A-4461-BD5E-FE0611BA102D}
HashSha1 **HashSha1Mt**	{11056249-400A-4461-BD5E-FE06113A1025} {11056249-400A-4461-BD5E-FE0611BA1025}
HashSha256 **HashSha256Mt**	{11056249-400A-4461-BD5E-FE06113A1026} {11056249-400A-4461-BD5E-FE0611BA1026}
HashSha384 **HashSha384Mt**	{11056249-400A-4461-BD5E-FE06113A1027} {11056249-400A-4461-BD5E-FE0611BA1027}
HashSha512 **HashSha512Mt**	{11056249-400A-4461-BD5E-FE06113A1028} {11056249-400A-4461-BD5E-FE0611BA1028}

#4.1 Ciphers

Block ciphers encrypt data in portions called 'blocks'; a typical block size is 64, 128, or 256 bits. Block ciphers are symmetric, that is they use the same key for encryption and decryption.

Block ciphers is the workhorse of cryptography and are used almost everywhere. IDcypher provides quite a number of interfaces you can use to access block ciphers - from the simplest single block encryption to complex high level stream processors.

IBlockCipherParams

IBlockCipherParams, which inherits ICipherParams methods, gets or sets block cipher parameters. You will rarely (or never) need to change cipher parameters, except ICipherParams.Key, which must be set before you start encryption or decryption. Just set it to any non-empty sequence of bytes and the cipher will normalize it transparently. You will need exactly the same sequence of bytes in order to decrypt your data.

If you have a password, don't use it as a key; use a hash function (see Hash Functions) or the Crypto.ConvertPassword function to convert it to a binary key as shown below:

Example

Visual Basic

```
Dim cipher As Object = New CipherBlowfish()
CType(cipher, ICipherParams).Key = Crypto.ConvertPassword("My Password", New
HashSha512(), Nothing)
```

C#

```
object cipher = new CipherBlowfish();
((ICipherParams)cipher).Key = Crypto.ConvertPassword("My Password", new HashSha512(),
null);
```

Keep in mind that the key must be kept safe; if the key is lost, the encrypted data is lost too because there is no known way to decrypt it without the key.

You might have also noticed the IBlockCipherParams.InitVector property. It sets a so-called initialization vector that is used to prevent a known plaintext attack on the cipher. If you use the same key for several streams, encrypt them using different initialization vectors. Although you can consider the initialization vector as an extension of the key, it is not secret and can be stored in the open without weakening encryption.

IRawBlockCipher

IRawBlockCipher provides two cipher primitives for encrypting and decrypting a single block of data. This interface does not perform any block chaining - it just encrypts/decrypts a single block according to the cipher specifications.

IBlockCipher

IBlockCipher is more advanced than IRawBlockCipher - it can encrypt any number of blocks and performs block chaining to make encryption stronger.

In order to encrypt (decrypt) data using IBlockCipher call IBlockCipher.Init, then repeatedly call IBlockCipher.Encrypt (IBlockCipher.Decrypt) until all data is processed, and then call IBlockCipher.Done to finish processing. The resulting data have exactly the same size as the source data and replaces the original data in the buffer. However the data must be provided as an integer number of blocks - for example, if the cipher block size is 128 bits, you can encrypt 16, 32, 48, 64, and so on bytes; attempt to encrypt, say, 20 bytes will result trhowing a CsException with "Invalid data size" error message.

ICipher

ICipher is the most commonly used low-level encryption interface. It is very similar to IBlockCipher, but it can process any number of bytes. ICipher prefixes the source stream with a random header to strengthen encryption, and pads it to make the stream length a multiple of cipher blocks.

As a result, the size of the output stream is always larger by a random number of bytes and cannot be stored to the same buffer. The solution is a callback function that is called every time the cipher have a block ready for output. The example below illustrates using ICipher for encrypting a stream.

Example

Visual Basic

```
Dim outstr As FileStream

'Callback function to write out encrypted data
Sub WriteDataCallback(ByVal sender As Object, ByVal buffer As Byte(), ByVal start As Integer,
ByVal size As Integer)
    outstr.Write(buffer, start, size)
End Sub

'Function reads input stream, encrypts it, and writes
'  the encrypted data into output stream
Sub EncryptStream(ByVal inStream As FileStream, ByVal outStream As FileStream)
    outstr = outStream

    'Create cipher and set key
    Dim bf As CipherBlowfish = New CipherBlowfish()
    bf.Key = New Byte() {1, 2, 3, 4, 5, 6, 7, 8, 9}

    Dim fsize As Long = inStream.Length
    Dim buf(65535) As Byte
    Dim len As Integer

    'Init encryption
    CType(bf, ICipher).Init(AddressOf WriteDataCallback)

    'Encrypt stream
    While fsize > 0
        len = CInt(Math.Min(fsize, buf.Length))
        inStream.Read(buf, 0, len)
        CType(bf, ICipher).Encrypt(buf, 0, len)
        fsize -= len
    End While

    CType(bf, ICipher).Done()
End Sub
```

C#

```
class StreamEncryptor {
  private Stream outstr;

  // Callback function to write out encrypted data
  private void WriteDataCallback(object sender, byte[] buffer, int start, int size) {
    outstr.Write(buffer, start, size);
  }

  // Function reads input stream, encrypts it, and writes
  // the encrypted data into output stream
  public void EncryptStream(Stream inStream, Stream outStream) {
    outstr = outStream;

    // Set key
    CipherBlowfish bf = new CipherBlowfish();
    bf.Key = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    // Create I/O buffer
    byte[] buf = new byte[64 * 1024];
    long streamSize = inStream.Length;
    int len;

    // Init encryption
    ICipher cipher = (ICipher)bf;
    cipher.Init(WriteDataCallback);

    // Encrypt stream
    while (streamSize > 0) {
      len = (int)Math.Min(streamSize, buf.Length);
      inStream.Read(buf, 0, len);
      cipher.Encrypt(buf, 0, len);
      streamSize -= len;
    }
    cipher.Done();
  }
}
```

IMemoryBlockCipher

IMemoryBlockCipher is designated for encrypting or decrypting a single memory block or a string. This interface is a wrapper around a powerful but more complex ICipher, simplifying working with memory-resident data.

IEncryptor

IEncryptor looks very similar to ICipher except it does not have Decrypt member and its Init method looks differently.

Its first argument is a compressor. If you set it to a valid compressor object, the input stream will be compressed before encryption. The code below illustrates the technique:

Example

Visual Basic

```
Dim zip As CompressorZip = New CompressorZip()
zip.Level = Crypto.MaxCompression
CType(cipher, IEncryptor).Init(zip, ...
```

C#

```
CompressorZip zip = new CompressorZip();
zip.Level = Crypto.MaxCompression;
((IEncryptor)cipher).Init(zip, ...
```

You can set this argument to null to turn off compression.

The second argument must be either null, or a valid hash function object. If you specify a hash function, IEncryptor will compute and store the header and stream checksums.

The recommended value for the keyVerification argument is Crypto.DefaultKeyVerification.

Note that you don't have to specify an initialization vector - IEncryptor creates and stores it transparently.

You can refer to the example application FEncr, which encrypts files using IEncryptor.

IDecryptor

IDecryptor complements IEncryptor and decrypts the encrypted stream created by it. Using IDecryptor is quite obvious; you can also refer to the example application FDecr that decrypts files created by FEncr.

#4.2 Hash Functions

Hash functions, sometimes called checksum or message digest functions, convert a block of data (which may be very large) to a short fixed-size value. There is virtually zero probability of different data blocks having the same hash; besides, even a smallest modification of the source data will produce a different hash. This makes hash functions invaluable when you need to check the data for integrity and authencity.

The IDcypher hash function components provides three interfaces:

IHashFunctionParams

IHashFunctionParams gets or sets various hash function parameters, like the number of hashing passes, or the size of the resulting hash value. You won't need this interface often though, because most hash functions don't allow changing these parameters.

IMemoryBlockHash

IMemoryBlockHash contains two method: IMemoryBlockHash.HashBlock for hashing an array of bytes, and IMemoryBlockHash.HashString for computing a hash of a text string. You will find it very convenient for hashing small blocks, for example, finding a SHA-256 hash of the text string "qwerty" will look as

Example

Visual Basic

```
'Create SHA-256 hash function
Dim hf As IMemoryBlockHash = new HashSha256()
'Compute hash
Dim hashval As Byte() = hf.HashString("qwerty")
```

C#

```
// Create SHA-256 hash function
IMemoryBlockHash hf = new HashSha256();
// Compute hash
byte[] hashval = hf.HashString("qwerty");
```

IHashFunction

Sometimes data cannot be accessed as a single block. For example, reading a large file as a whole into memory is not always possible, and usually not a good idea even when possible.

For large volumes of data the IHashFunction interface provides a more flexible way of hashing data.

Using it only slightly more complex than using IMemoryBlockHash - first call IHashFunction.Init to start hashing, then call the IHashFunction.Hash method for every data chunk, and when all data are processed, call IHashFunction.Done to finish hashing and obtain the resulting hash value.

#5 Usage Notes

#5.1 Static vs. Dynamic Linking

There are two ways of instantiating a IDcypher component class.

Static Linking

First is the usual way of creating a class instance by using the new keyword, for example

Example

Visual Basic

```
Dim hashFunc As Object = New HashSha256()
```

C#

```
object hashFunc = new HashSha256();
```

In order to link statically, you should include all the required components into the References list.

Dynamic Linking

Another way to create the same component is to call the CsComponents.CreateComponent static function with the component Guid:

Example

Visual Basic

```
Dim hashFunc As Object = CsComponents.CreateComponent(Crypto.HashSha256)
```

C#

```
object hashFunc = CsComponents.CreateComponent(Crypto.HashSha256);
```

In this case you should reference only the root IDcypher.dll assembly which contains the dynamic loading code.

Which Method To Use?

Using the new keyword works best for a project that references a strictly defined set of components.

The CsComponents.CreateComponent function performs dynamic loading and so works slightly slower. On the other hand, you don't need to add a reference to the component to your project. In order to add and use a new component you need only to copy it into the application directory.

Another and most important adavantage of dynamic loading is its ability to load compatible components. For example, if you try to instantiate a non-existent Windows shredder, CreateComponent will try loading the standard multiplatform shredder. If you try loading a parallelized cipher on a single-processor system wich does not include parallelized components, CreateComponent will transparently loads the standard single-thread version.

Full Package

IDcypher also includes the IDcypherFull.dll assembly, which contains all the IDcypher components. If you don't need compatible component loading (or dynamic loading in general), and if you don't care about small disk space overhead, just include IDcypherFull.dll into the application's References list.

Note that even if dynamic loading has little sense with IDcypherFull.dll, it will nonetheless work perfectly. A program that worked with the component-based IDcypher, will work with the full-package IDcypher without any modification and vice versa.

#6 Copyright Information

```
#region Copyright (c) 2018 TRUST 47 Fund. All Rights Reserved.
//-------------------------------------------------------------------------//
//                           TRUST 47 Fund                                 //
//             Copyright (c) 2018 All Rights reserved                      //
//                                                                         //
//                                                                         //
// This file and its contents are protected by United States and           //
// International copyright laws.  Unauthorized reproduction and/or        //
// distribution of all or any portion of the code contained herein         //
// is strictly prohibited and will result in severe civil and criminal     //
// penalties.  Any violations of this copyright will be prosecuted         //
// to the fullest extent possible under law.                               //
//                                                                         //
// THE SOURCE CODE CONTAINED HEREIN AND IN RELATED FILES IS PROVIDED       //
// TO THE REGISTERED DEVELOPER FOR THE PURPOSES OF EDUCATION AND           //
// TROUBLESHOOTING. UNDER NO CIRCUMSTANCES MAY ANY PORTION OF THE SOURCE   //
// CODE BE DISTRIBUTED, DISCLOSED OR OTHERWISE MADE AVAILABLE TO ANY       //
// THIRD PARTY WITHOUT THE EXPRESS WRITTEN CONSENT OF TRUST 47 Fund        //
//                                                                         //
// UNDER NO CIRCUMSTANCES MAY THE SOURCE CODE BE USED IN WHOLE OR IN       //
// PART, AS THE BASIS FOR CREATING A PRODUCT THAT PROVIDES THE SAME, OR    //
// SUBSTANTIALLY THE SAME, FUNCTIONALITY AS ANY TRUST 47 Fund PRODUCT.     //
//                                                                         //
// THE REGISTERED DEVELOPER ACKNOWLEDGES THAT THIS SOURCE CODE             //
// CONTAINS VALUABLE AND PROPRIETARY TRADE SECRETS OF TRUST 47 Fund        //
// THE REGISTERED DEVELOPER AGREES TO EXPEND EVERY EFFORT TO               //
// INSURE ITS CONFIDENTIALITY.                                             //
//                                                                         //
// THE END USER LICENSE AGREEMENT (EULA) ACCOMPANYING THE PRODUCT          //
// PERMITS THE REGISTERED DEVELOPER TO REDISTRIBUTE THE PRODUCT IN         //
// EXECUTABLE FORM ONLY IN SUPPORT OF APPLICATIONS WRITTEN USING           //
// THE PRODUCT.  IT DOES NOT PROVIDE ANY RIGHTS REGARDING THE              //
// SOURCE CODE CONTAINED HEREIN.                                           //
//                                                                         //
// THIS COPYRIGHT NOTICE MAY NOT BE REMOVED FROM THIS FILE.                //
// ----------------------------------------------------------------------- //
#endregion Copyright (c) 2018 TRUST 47 Fund. All Rights Reserved.
```

#7 Warranty Disclaimer

A **Warranty Disclaimer** clause in a software license disclaimer clause repudiates all warranties not expressly provided. In some instances, the licensor will provide limited warranties for media, software and support. In other cases, no warranties are given, the software is provided "as-is" with all faults.

NO WARRANTY
MEDIACRYPT.COM DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE AND THE ACCOMPANYING WRITTEN MATERIALS. NO LIABILITY FOR CONSEQUENTIAL DAMAGES.
IN NO EVENT SHALL MEDIACRYPT.COM OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THIS MEDIACRYPT/IT PRODUCT, EVEN IF MEDIACRYPT.COM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

=== End Of Document ===