maXbox

maXbox Starter 13

Start with Crypto Programming

1.1 Set a Cipher

Today we spend another suspense time in programming with cryptography and some stories about it. CryptoBox, which is now included in maXbox, is a comprehensive, free e-learning app about cryptography. The relationship of the topics cryptology, IT security, data protection and risk management is described here and we're going to take a look at the code too.

CryptoBox is based on Turbo Power LockBox 3 which is a big Delphi library for cryptography. LockBox 3 is a FOSS Delphi Cryptographic Library, providing efficient private key encryption, public key encryption and hashing functions. Currently supported in Delphi 7, 2005, 2007, 2009 and 2010. It provides support for AES, DES, 3DES, Blowfish, Twofish, SHA, MD5, a variety of chaining modes, RSA digital signature and verifications.
We're just going to explain AES with the new SHA-512/256 hashes.

Hope you did already work with the Starter 1 to 12 available at:

http://www.softwareschule.ch/maxbox.htm

This lesson will introduce you to encryption and decryption. An encryption algorithm is required in order to transmit confidential information over insecure information channels, for example, over a network or a stick. The information is encrypted by the originator prior to transmission and decrypted by the recipient following transmission.

☞A symmetric encryption algorithm is one in which the originator's and recipient's keys are identical. Encryption algorithms in which the originator and recipient have different keys are called asymmetric.

Encryption is a wide topic with themes like analysis, protocols, passwords, randomness and individual procedures. For example cryptographic protocols aggregate cryptographic basis functions (called primitives) and are used between several participants. Therefore they define rules for the format, the content, the meaning and the order of exchanged messages.

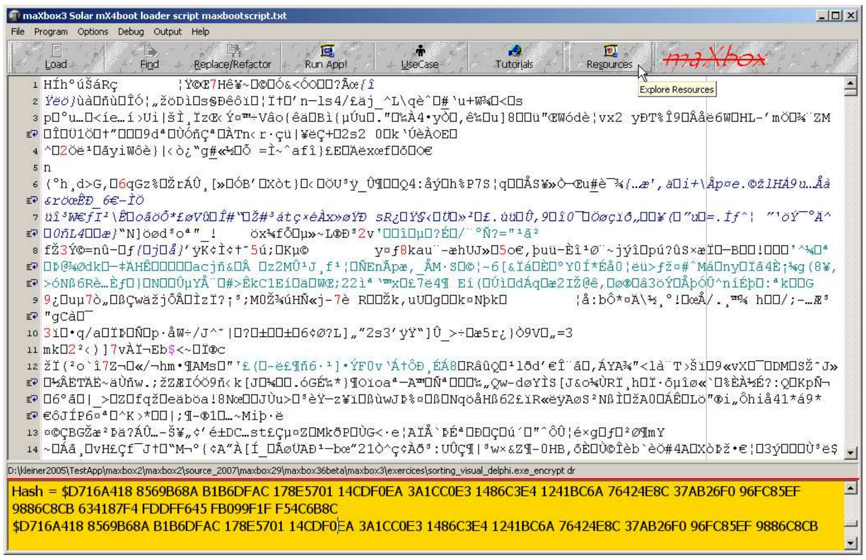
Let's begin with the application structure process in general of an encryption:

1. Set a password based on a rule
2. Let derive a key from the password (key length: AES 256)
3. Get data with to encrypt
    - Compute SHA hash (SHA 256)
    - Encrypt data with a key (session key)
4. Store the cipher on a medium.

☞All classic encryption methods belong to the group of symmetric encryption methods, where sender and receiver using the same key (or password), so AES belongs to it.

If the cipher data is to be used with visual data controls, you need a special (hex) editor to show the output of the cipher (also possible in maXbox, see below) but in our app we go straight with the protocol result to the output of the memo shell and write in a log file too.
The log file is defined in a const:

```
21 const    CIPHERLOG = 'cipherbox_log2.txt';
```


maXbox3 Solar mX4boot loader script maxbootscript.txt
File Program Options Debug Output Help
Load
Find
Replace/Refactor
Run App!
UseCase
Tutorials
Resources
maXbox
Explore Resources

```
D:\kleiner2005\TestApp\maxbox2\maxbox2\source_2007\maxbox29\maxbox36beta\maxbox3\exercices\sorting_visual_delphi.exe_encrypt dr
```

```
Hash = $D716A418 8569B68A B1B6DFAC 178E5701 14CDF0EA 3A1CC0E3 1486C3E4 1241BC6A 76424E8C 37AB26F0 96FC85EF
9886C8CB 634187F4 FDDFF645 FB099F1F F54C6B8C
$D716A418 8569B68A B1B6DFAC 178E5701 14CDF0EA 3A1CC0E3 1486C3E4 1241BC6A 76424E8C 37AB26F0 96FC85EF 9886C8CB
```

1: Cipher Result as of ASCII with Hash output

1.2 Code with AES

As you already know the tool is split up into the toolbar across the top, the editor or code part in the centre and the output window shell like a crypto protocol result at the bottom.
Included in the package maXbox is the integrated encryption utility CryptoBox as an add-on listed on the menu `/Options`. But we show the tool as a script with source, an add-on is more a compiled and integrated script.

Before this starter code will work you will need to download maXbox from the website. It can be down-loaded from http://sourceforge.net/projects/maxbox site. Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox3.exe` the box opens a default program. Make sure the version is at least 3.8 because LockBox need that. Test it with F2 / F9 or press **Compile** and you should hear a sound a browser will open. So far so good now we'll open the example:

```
258_AES_cryptobox2.txt
```

If you can't find the file use the link (size 10 KB, 10120 Bytes):

http://www.softwareschule.ch/examples/258_AES_cryptobox2.txt

Or you use the `Save Page as…` function of your browser[1] and load it from `examples` (or wherever you stored it). One important thing: You can't destroy your primary file because the script and tool always makes an encrypted or decrypted copy of your file. So the encrypted file is ready to send or store in another medium. It's up to you to delete (better DOD wipe) the original file after encrypting but on you own risk.
So what are the risks? You can forget the password or the process could be interrupted resulting in a damaged file. So you can restore your file only in relation to the known password or with a backup.
Now let's take a look at the code of this project first with the password set. Our first line is

```
15 Program CipherBox3_FORM_Lab;
```

[1] Or copy & paste

We have to name the program it's called `CipherBox3_FORM_Lab`. Next we jump to line 68:

```
67 begin
68  AESpassw:= InputBox('AES CryptoBox Password','Enter Password:','password');
```

First in line 68 we ask to set a password. Passwords are important for all IT systems that offer interfaces for user access. Particularly, this includes machines in the private and commercial sector as well as services like e-mail or online banking. In these cases the logon is usually done by entering a user name and the according password.

A secure password is a password that cannot easily be guessed. You can access this dialog by pushing a password guideline button in the dialog password.

You do not need to memorise a crypt password like X34RTG11)@j, use a pass phrase or a pass sentence to avoid loosing your memory;). Ex. `054_pas_speakpassword2.txt` deploys speak able passwords with the possibility to add one or more special signs at the end:

```
Result: my password is: xetesose&1
```

The secure storage of passwords also plays a big role for password security. On the one hand, this means one should never write passwords down in order to prevent unauthorized access. On the other hand, this includes technical measures like hash functions when passwords are transmitted over public networks or stored in password files.

The size of the password space is the amount of passwords that can be created with all available characters (the alphabet). Assume you have an alphabet of 52 letters (upper- and lowercase) as well as ten digits, then the password space consists of 62^8, that is around 218 trillion passwords.
In statistic we call this a permutation which means arrangement of things. The word arrangement is used, if the order of things is considered but in our case it's a variation (permutation with repeating) so we compute = `n^k`.
If you increase the size of the alphabet with 10 more characters (for example special characters), the password space grows to more than three times of its size. The bigger the space, the better.

```
Result:= Writeln(FloatToStr(IntPower(62,8))) = 218340105584896
```

This crypto app requires also a hash with objects of the classes: `TSHA2` and `TAES` and several predefined methods. The user interface shall be clean and simple. For Ciphers and Hashes, two styles shall be provided: A component and an interface pointer.
So what's a hash?

A hash function is a (mathematical) function which receives an input of arbitrary length and returns a function value (hash value) of a <u>fixed length</u> (usually 128 or 160 bits). The hash functions which are used in cryptography should be called one-way hash functions.

```
Source: CrypTool 1.4.30
```

The hash function itself must be public, i.e. everyone should be able to (efficiently) calculate the hash value for a given input. Conversely, however, it must be (computationally) infeasible to find an input for a given value which possesses exactly the predetermined value as hash value (this is referred to as a one-way characteristic).

The hash value `H = H(M)` of an original message M, is used to check, if the received message M' has been changed or not.
In line 102 we compute such a hash:

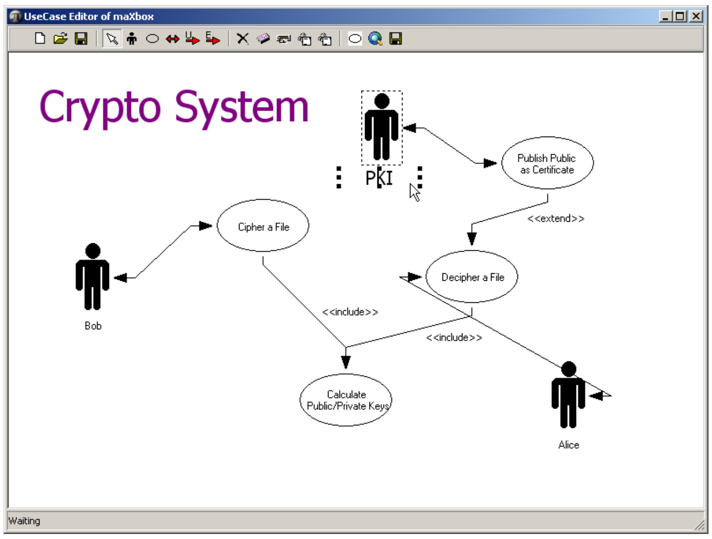
```
 102      Add('is: '+ComputeSHA256(selectFile,'F'));
```

The parameter F stands for File and the parameter S for String. With `ComputeSHA512()` we double the result size of a hash:

Hash 256 = $65B8B635 FEFEF4B6 CDE381AD 1ECC15E2 3E6092CC D1B630D9 B726A8ED E94CB28B

Hash 512 = $B70D7466 0A99BE83 7544F2A7 E1D62916 C201F410 B9DD6668 C8F92B4F 31C33A03 DEFC3A88 752EC22F E6634DA5 B9D8CC52 23914739 11CD3D7C 8656B250 41E0A7E8

The length of the output (hash value) is always between 16 bytes and 64 bytes (between 128 bit and 512 bit) long - depending on the hash method.



Till now we are discussing the topics of password and hashes therefore we come now to the base function of a crypto system. As I said earlier we use AES with password.

1.3 The winner is AES

NIST (U.S. National Institute of Standards and Technology) set up an international competition with the goal to create a new symmetric encryption algorithm that should be come a new standard in the USA. The competition ended on 2 October 2000 when Norman Y. Mineta declared Rijndael as the new national standard.
So Rijndael is the winner of the three year competition where a number of the most well known cryptographers were involved. Mineta said "finally this standard will serve as a tool for security critical IT-applications. Rijndael enables eCommerce and eGovernment and will create new opportunities."
Rijndael was developed by the Belgian cryptographers Joan Daemen from Proton-Welt International and Vincent Rijmen, member of the Catholic University Leuven. Both of them are known as very experienced members of the crypto community. `Source: CrypTool 1.4.30`
After the final acceptance AES will be a publicly available patent free standard used to protect long-term data. It will substitute the DES-Algorithm (Data Encryption Standard) which was introduced in 1977.
AES is used to exchange encrypted data with other users. It does not contain functions for key management. The keys have to be exchanged between the users on a secure channel. In our case we just use a secure password!

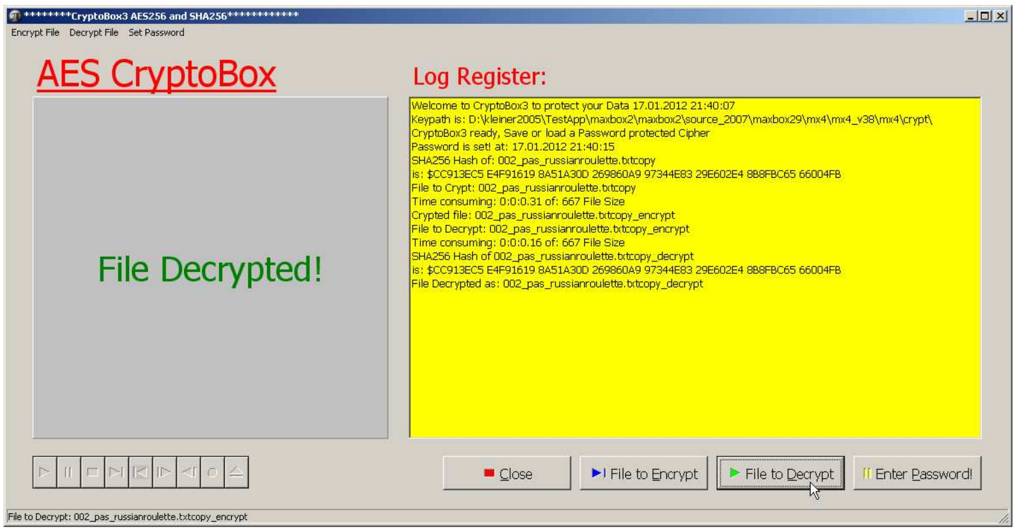
```
procedure EncryptMediaAES(sender: TObject);


 with TStopwatch.Create do begin
    Start;
    AESSymetricExecute(selectFile, selectFile+'_encrypt',AESpassw);
    mpan.font.color:= clblue;
    mpan.font.size:= 30;
    mpan.caption:= 'File Encrypted!';
    Screen.Cursor:= crDefault;
    Stop;
    clstBox.Items.Add('Time consuming: ' +GetValueStr +' of: '+
          inttoStr(getFileSize(selectFile))+' File Size');
    Free;
  end;
```

And that's how to get a feeling of the speed of AES with a protocol extract:

```
Use native.AES-256 cipher and native.CBC chaining mode.
Ciphertext size = 21726 bytes.
Decryption succeeded. 21726 bytes processed.
Speed of decryption was 1326 KiB per second.  //4 GByte about 40 Minutes
```

The AES statement in our case is a command that contains hard-coded extension names and values, or it can be a parameterized with a configuration that contains replaceable parameters that represent field values that must be bound into the statement before it is executed.
The spirit of LockBox is its intended offering of features (AES, DES, 3DES, and various chaining modes, MD5, SHA-1 and RSA).
AES specifies a cryptographic algorithm that can be used to protect electronic data by encrypting (enciphering) and decrypting (deciphering) information.



```
Welcome to CryptoBox3 to protect your Data 17.01.2012 21:40:07
Keypath is: D:\kleiner2005\TestApp\maxbox2\maxbox2\source_2007\maxbox29\mx4\mx4_v38\mx4\crypt\
CryptoBox3 ready, Save or load a Password protected Cipher
Password is set! at: 17.01.2012 21:40:15
SHA256 Hash of: 002_pas_russianroulette.txtcopy
is: $CC913EC5 E4F91619 8A51A30D 269860A9 97344E83 29E602E4 8B8FBC65 66004FB
File to Crypt: 002_pas_russianroulette.txtcopy
Time consuming: 0:0:0.31 of: 667 File Size
Crypted file: 002_pas_russianroulette.txtcopy_encrypt
File to Decrypt: 002_pas_russianroulette.txtcopy_encrypt
Time consuming: 0:0:0.16 of: 667 File Size
SHA256 Hash of 002_pas_russianroulette.txtcopy_decrypt
is: $CC913EC5 E4F91619 8A51A30D 269860A9 97344E83 29E602E4 8B8FBC65 66004FB
File Decrypted as: 002_pas_russianroulette.txtcopy_decrypt
```

File to Decrypt: 002_pas_russianroulette.txtcopy_encrypt

2: AES Box of the GUI with Log

Hard-coded statements are useful when applications execute exact, known values or extensions each time they run. At design time or runtime you can easily replace one hardcode query with another hard-coded or parameterized statement as needed.
For example, we use the following hard-coded specification statements:

```
1. native.AES-256;
2. AKEYSIZEBIT = 2048;
3. native.CBC chaining mode;
4. opCustomBlockSymetricEncrypt
5. little-endian[2] hex SHA Format rendering
```

So another way to explain a block cipher is to discuss the last block of a cipher. The plaintext is divided into blocks of equal size (one block consists of as many characters as the key has rows).

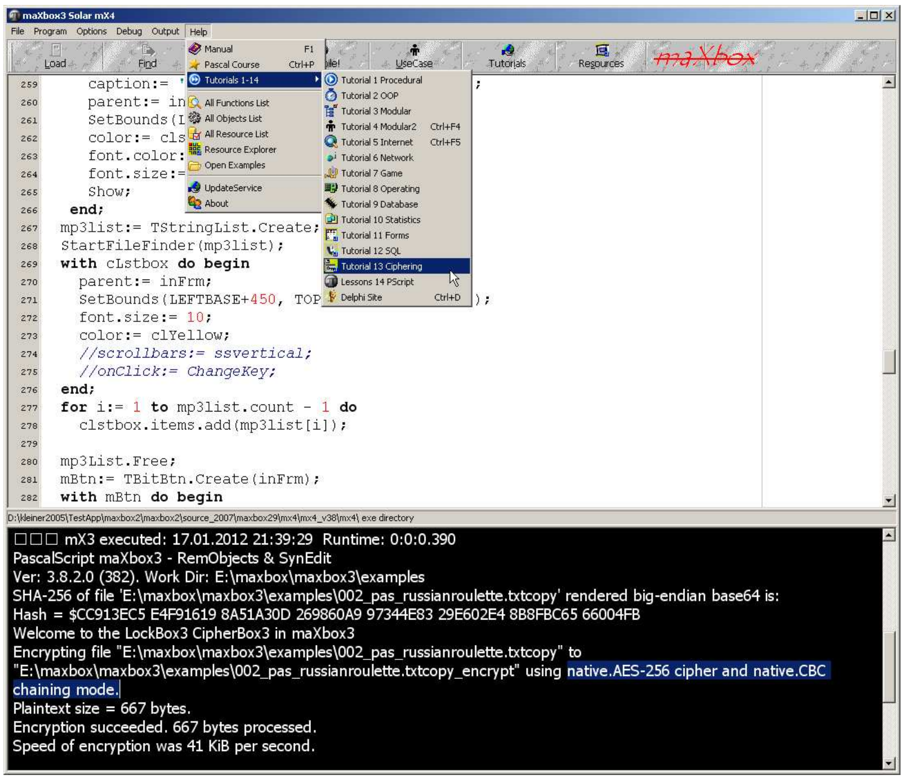
[2] **Big Endian** the high-order byte of the number is stored in memory at the lowest address, and the low-order byte at the highest.

The last block sometimes has too few characters: In this case the last block is padded to have the length of the matrix dimension.
It was a requirement that the new algorithm (AES) must satisfy a symmetric block cipher but also should or must be usable for Stream Cipher, Message Authentication Code (MAC) generator, pseudorandom number generator, hash function etc.

Now we jump to the decryption:

```
begin
with TStopwatch.Create do begin
    Start;
    AESDecryptFile(selectFile, selectFile+'_decrypt',AESpassw);
    clearout:= selectFile+'_decrypt';
    Delete(clearout, length(clearout)-15, 8); //-7!
    RenameFile(selectFile+'_decrypt', clearout);
    Screen.Cursor:= crDefault;
    Stop;
    clstBox.Items.Add('Time consuming: ' +GetValueStr +' of: '+
          inttoStr(getFileSize(selectFile))+' File Size');
    Free;
 end;
```

If a password is set and a file with the extension `_decrypt` is found, the command `AESDecryptFile` executes the statement when you call the `OpenFileDialog` of the procedure `PromptForFileName` behind the button <`File to Decrypt`>. If the statement does not return a result set then a password is not set. The `RenameFile()` function renames old name file or directory to new name, returning true if successful, so we just add a new extension in exchange.
If a file or directory name is given without a path, the file must be in the current directory.



```
      caption:= '
      parent:= in
      SetBounds(I
      color:= cls
      font.color:
      font.size:=
      Show;
    end;
  mp3list:= TStringList.Create;
  StartFileFinder(mp3list);
  with cLstbox do begin
    parent:= inFrm;
    SetBounds(LEFTBASE+450, TOP
    font.size:= 10;
    color:= clYellow;
    //scrollbars:= ssvertical;
    //onClick:= ChangeKey;
  end;
  for i:= 1 to mp3list.count - 1 do
    clstbox.items.add(mp3list[i]);

  mp3List.Free;
  mBtn:= TBitBtn.Create(inFrm);
  with mBtn do begin
```

```
□□□ mX3 executed: 17.01.2012 21:39:29  Runtime: 0:0:0.390
PascalScript maXbox3 - RemObjects & SynEdit
Ver: 3.8.2.0 (382). Work Dir: E:\maxbox\maxbox3\examples
SHA-256 of file 'E:\maxbox\maxbox3\examples\002_pas_russianroulette.txtcopy' rendered big-endian base64 is:
Hash = $CC913EC5 E4F91619 8A51A30D 269860A9 97344E83 29E602E4 8B8FBC65 66004FB
Welcome to the LockBox3 CipherBox3 in maXbox3
Encrypting file "E:\maxbox\maxbox3\examples\002_pas_russianroulette.txtcopy" to
"E:\maxbox\maxbox3\examples\002_pas_russianroulette.txtcopy_encrypt" using native.AES-256 cipher and native.CBC
chaining mode.
Plaintext size = 667 bytes.
Encryption succeeded. 667 bytes processed.
Speed of encryption was 41 KiB per second.
```

3: The Result of the Script from the protocol

And you can also use the methods of symmetric cipher to a hybrid crypto system!

The hybrid encryption system applies to an active document a symmetric cipher (here AES) using a session key. The session key is then encrypted with an asymmetric cipher (mostly RSA).
At runtime, use also properties and methods of the LockBox library to set new current ciphers (see Appendix), add new files, or to compare existing contents by a predefined `SymetricCompareFiles` routine for example:

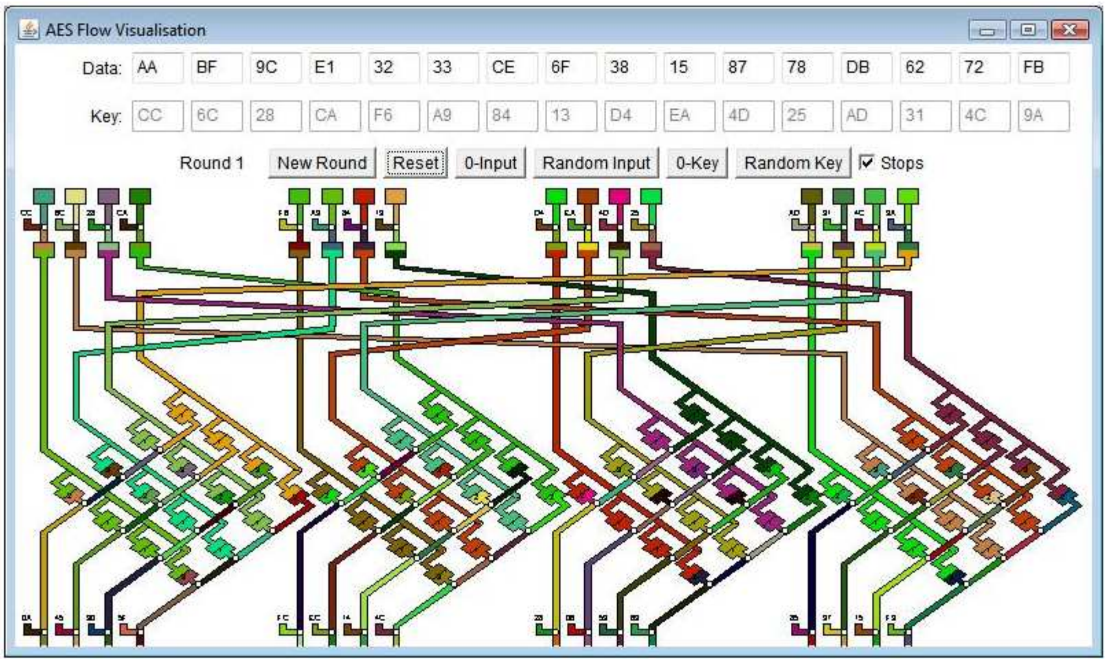
```
passw:='password1';
AESEncryptFile(exepath+'examples\citymax.bmp',exepath+'examples\cityc.enc1',passw);

AESDecryptFile(exepath+'examples\citymax2.bmp',exepath+'examples\cityc.enc1','passw
');  //direct call with hard coded filenames to explain

SymetricCompareFiles(exepath+'examples\citymax.bmp',exepath+'examples\citymax2.bmp'
);// to test the integrity
if
CompareFiles(exepath+'examples\citymax.bmp',exepath+'examples\citymax2.bmp',NIL,NIL
) then writeln('bitmap comp OK') else writeln('comp NOK');
```

Now we introduce a visualization of the AES algorithm.

The AES flow visualization of the tool CrypTool uses a colourful display to show how data changes in each round of the AES encryption algorithm. This is shown using the example of AES128 with 10 rounds, where data as well as the key consist of a 16 byte block (128 bits) each.
Every byte of the 16 byte block is illustrated by a colour stream, whereas the colour reflects the current value (through direct derivation of the RGB values from the numerical values of the bytes - see http://en.wikipedia.org/wiki/RGB_color_model).
This way you can easily see that a data block consisting of zeros only will turn "coloured" very fast, even though the key consists of zeros only, too.



Data:	AA	BF	9C	E1	32	33	CE	6F	38	15	87	78	DB	62	72	FB
Key:	CC	6C	28	CA	F6	A9	84	13	D4	EA	4D	25	AD	31	4C	9A

You can enter each byte manually - this is applies to both, the data block as well as the key block. Although it is nice to get the information on how many rounds are in a matrix, calculating the `NewRound` itself is very hard to understand.
The overall aim is to produce a maximum of entropy.

The entropy of a document is an index of its information content. This means that the information content depends exclusively on the probability distribution with which the source generates the messages. Let's say that if each sign has the same probability it results in maximum entropy.
Entropy is an expression of insecurity as the number of Yes/No questions which have to be answered in order to clarify a message or a character. If a character has a very high probability of occurrence, then its information content is low. This would be the case, for example, with a business partner who regularly replies "Yes". This reply also does not permit any conclusions to be drawn as to understanding or attention. Replies which occur very seldom have high information content.

Therefore in a document in which all 26 characters of the set [A..Z] occurs equally often we get the maximum entropy of 4.70 Bit!

```
     (Writeln(floatToStr(Log2(26))) = 4.70043971814109.
```

We come now to the main section:

```
begin
      SetCryptFormAES;
      if DirectoryExists(ExePath+KEYPATH) = false then
        CreateDir(ExePath+ KEYPATH);
      cryptLog:= ExePath+ CIPHERLOG;
      writeln('Machine: '+GetHostName)
      writeln('User: '+GetUserName)
      writeln('Instance: '+intToStr(getHinstance))
      writeln('Procid: '+intToStr(getProcessid))
      writeln('Processors: '+intToStr(GetNumberOfProcessors))
      WriteLog(cryptLog, memo2.text)
End.
```

A main section behaves in many ways very much like a loop or sequence, except that you use a Form which is event-driven. First the form is created and then some information is written in a log or output. Some features are still missing:

- Remembers its main window position and size and uses this when it is restarted.
- Add a function of entropy of clear text, clear text and cipher text.
- Using win it is possible to get the online help via F1.
- Build with REGEX a password policy checker.
- Safeguarding entry fields against fuzzy input via clipboard, to avoid provoked crashes.

Finally, SHA2 family (SHA-256, SHA-512) is added for calculating the hash value of the shown document or of a file. The Rijndael encryption algorithm works with a variable block length e.g. 128 bits, and a variable key length e.g. 128, 192 and 256 bits.
CryptoBox uses Rijndael in CBC mode with zero initialization vectors and 01-00 padding.

The funny cartoon "A Stick Figure Guide to the Advanced Encryption Standard (AES)" by Jeff Moser, published in September 2009, explains the AES algorithm in 4 acts:

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

Password or Keyfile CryptoBox does not allow recovery of any encrypted data without knowing the correct password or key. We cannot recover your data because we do not know and cannot determine the password you chose or the key you generated using AES. The only way to recover your files is to try to "crack" the password or the key, but it could take thousands or millions of years (depending on the length and quality of the password or key files, on the software/hardware performance, algorithms, and other factors).
Note: Modern block ciphers encrypt by mixing data over a series of rounds to produce a complex nonlinear relationship between the plaintext (message), cipher text and key. For AES-128, AES-192

and AES-256 the number of rounds is 10, 12 and 14 respectively. So AES-256 requires a further 4 rounds over AES-128 to mix the additional key bits, and is consequently only 40% slower.

On average the correct key will be recovered after `2^(n-1)` guesses, or searching half the key space. So AES-256 should be secure against an attacker who has materially less than `2^(255) = 10^(76)` resources at their disposal. Well, unless you have a need to approximate the number of hydrogen atoms in the observable universe, you will never encounter a number as large as `10^(76)`. So there is little hope in attacking AES-256 by direct brute force methods, but is this the only way to get at the data protected by such large keys?
It is common for encryption solutions to use one of the following key management systems:

- Derive the device-encryption key directly from a user-supplied password, using a method such as the Password-Based Key Derivation Function.
- Randomly generate the device-encryption key, and then encrypt this key by a key derived from a user-supplied password

In this tutorial we have considered two cases where this can happen: long keys being protected by weaker passwords, and long keys being protected by much shorter (weaker) keys. The moral seems clear: don't use very long keys in a given environment unless the security of the other parameters involved is similar, since the weakest link will be the target!
Tasks for advanced studies:

📖 Study modern (asymmetric) crypto methods and protocols (hash, RSA, ECC, digital signature, hybrid encryption, PKCS#5, shared secret, ...) and partly attacks against it (birthday attack, side-channel attack, lattice-base reduction).

📖 Read the Chinese Labyrinth by Dr. Carsten Elsner about Primes.

📖 Study all about a salt. A 512-bit salt is used by password derived keys, which means there are 2^512 keys for each password. So a same password doesn’t generate always a same key. This significantly decreases vulnerability to 'off-line' dictionary' attacks (pre-computing all keys for a dictionary is very difficult when a salt is used). The derived key is returned as a hex or base64 encoded string. The salt must be a string that uses the same encoding.

We also use a lot of more crypto scripts to teach, simply take a look in the meantime at `210_public_private_cryptosystem.txt.` At least let’s say a few words about random generation and what functions they perform.

4: A real LockBox

☝ The many applications of randomness have led to the development of several different methods for generating random data.

Tools implement a number of different pseudorandom number generators. For the generators listed below in CrypTool parameters can be adjusted.

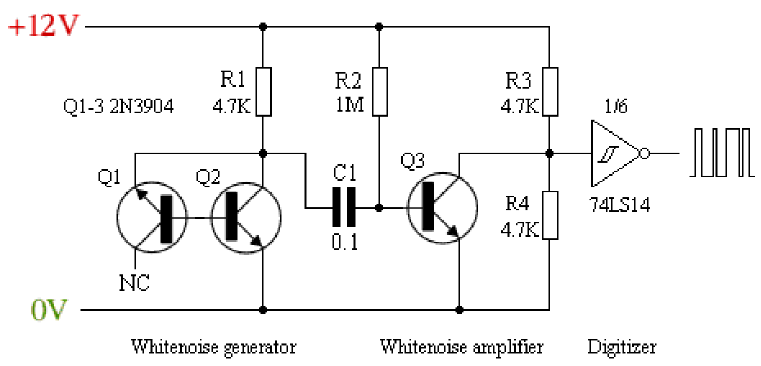
- The random generator provided by Secude
- X^2 modulo N Generator
- Linear Congruence Generator
- Inverse Congruence Generator

A random seed (or seed state, or just seed) is a number (or vector) used to initialize a pseudorandom number generator. The choice of a good random seed is crucial in the field of computer security. When a secret encryption key is pseudo randomly generated, having the seed will allow one to obtain the key.
The quality of the output generated by the random number generator can be tested with the aid of the random tests, periodicity analysis, compressibility and the Vitányi analysis implemented in CrypTool.

☞ So far we have learned something about encryption coding, protecting and cracking, let's make a conclusion to the structure at the beginning of our demo:

The main problem of all symmetric encryption algorithms is the key management for a large number of secret keys (one for each pair of communicating parties). Asymmetric cryptography solves this problem. Symmetric algorithms on the other hand are much faster than the asymmetric ones.
`Source: CrypTool 1.4.30`

Asymmetric encryption systems are, due to their large computational requirements, used in practice only for encrypting so called "session keys". The pay load is encrypted with a symmetric algorithm using the session key. This combination of two methods is called hybrid encryption (asymmetric/symmetric). This implies no security deficit if you use a good random number generator for the session key, like a HW generator below!



Try to change the SHA size to SHA1 in order to compare:

```
04 Writeln(SHA1(exepath+'maxbox3.exe'));
```

Try to find out more about the Pascal random generator and the function of `Randomize`.

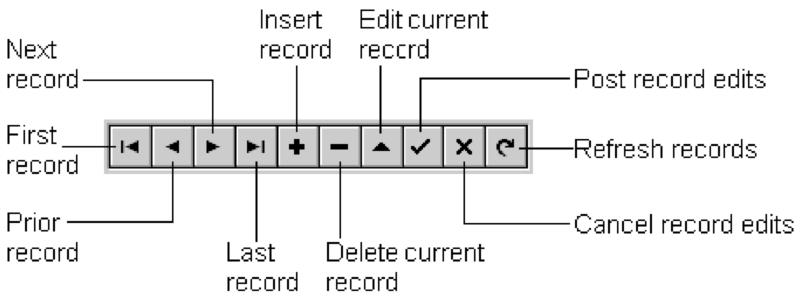
Check the source of LockBox to find out how a key is derived from a password or seed:

```
05 E:\maxbox\maxbox3\source\TPLockBoxrun\ciphers\uTPLb_AES.pas;
```

Try to change the exception message with a string literal const or resource line:

```
82 except
83   E:= Exception.Create('Password not Valid or too weak');
84   Showmessage(E.message+'No AES key has been produced')
85 end;
```

Case Study: Install (Code) a DB Navigator on the form as you can already see at the bottom left. A `DBNavigator` component enables the user to browse a dataset record by record. The navigator provides buttons for first record, next record, previous record, last record, insert record, delete record, edit record, cancel edits, post edits, and refresh.
The case study should answer the following question: Is it possible to cipher / decipher a file with this navigator. You can imagine a record as a block, a file or a whole partition on a hard disk and we transform this in a block cipher step by step.



max@kleiner.com
Links of maXbox and Cryptography Resources:

http://www.softwareschule.ch/maxbox.htm
http://sourceforge.net/projects/maxbox
http://sourceforge.net/apps/mediawiki/maxbox/

CrypTool is a comprehensive, free e-learning program about cryptography and cryptanalysis (cryptology):
http://www.cryptool.com

My Own Experience:
http://www.softwareschule.ch/download/armasuisse_components.pdf

SHA1 Hash of maXbox 3.8.2 Win: A47BD7EA26878DCEF2F74852DE688686A1D9523B

1.4 Appendix: AES Import Unit in maXbox

The shortest poem about cryptography is: Cry-Crypt-Crypto-Crypto logic.

Reference to AES:
http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf

This file is part of Turbo Power LockBox 3 and is called the import unit for maXbox. Turbo Power LockBox 3 is free software being offered under a dual licensing scheme: LGPL3 or MPL1.1.

- An OpenSSL wrapper for RSA functions, which include:
  - Generate key pair
  - Load/Save public/private keys in PEM format
  - Sign and Verify
  - Only works with a fairly recent version of libeay32.dll, namely, at least version 1.0.0.0 is required.
  - Demo program enhanced to include demo of the OpenSSL wrapper.
- Optional UTF-8 password for TCodec with UNICODE enabled compilers.
- Delphi XE, XE2 support
- Win 64 platform support
- Installer support extended for Delphi XE, XE2

Unit Import Implementation

```
uses
   uTPLb_BlockCipher
  ,uTPLb_StreamCipher
  ,uTPLb_Decorators
  ,uTPLb_AES
  ,uTPLb_StreamUtils
  ,uTPLb_Constants
  ,uTPLb_Codec
  ,uTPLb_CryptographicLibrary
  ,uTPLb_SHA2
  ,uTPLb_Hash
  ,fMain;

const
  AKEYSIZEBIT = 2048;
  //AKEYSIZEBIT = 1024;

  ChainIds: array[ 0..6 ] of string = (
   ECB_ProgId, CBC_ProgId, PCBC_ProgId, CFB_ProgId,
   CFB8bit_ProgId, CTR_ProgId, OFB_ProgId);

  CipherIds: array[ 0.. 8 ] of string = (
    'native.AES-128', 'native.AES-192', 'native.AES-256',
    'native.DES', 'native.3DES.1', 'native.3DES.2', 'native.Blowfish',
    'native.Twofish', 'native.XXTEA.Large.Littleend');

type
  TOperation = (opIdle, opSymetricEncrypt, opSymetricDecrypt,
      opSymetricCompare, opScribble, opCustomBlockSymetricEncrypt,
      opCustomBlockSymetricDecrypt, opRSAGen, opRSAEncrypt,
      opRSADecrypt, opSign, opVerify, opHash, opOpenSSL);

procedure SIRegister_uTPLb_AES(CL: TPSPascalCompiler);
begin
  SIRegister_TAES(CL);
  CL.AddDelphiFunction('procedure AESSymetricExecute(const plaintext, ciphertext, password: string)');
  CL.AddDelphiFunction('procedure AESEncryptFile(const plaintext, ciphertext, password: string)');
  CL.AddDelphiFunction('procedure AESDecryptFile(const replaintext, ciphertext, password: string)');
  CL.AddDelphiFunction('procedure AESEncryptString(const plaintext: string;
                                                  var ciphertext: string; password: string)');
  CL.AddDelphiFunction('procedure AESDecryptString(var plaintext: string;
                                                  const ciphertext: string; password: string)');

  CL.AddDelphiFunction('function ComputeSHA256(astr: string; amode: char): string)');
  CL.AddDelphiFunction('function ComputeSHA512(astr: string; amode: char): string)');
  CL.AddDelphiFunction('function SHA256(astr: string; amode: char): string)');
  CL.AddDelphiFunction('function SHA512(astr: string; amode: char): string)');

  CL.AddDelphiFunction('function EndianWord(w : word): word)');
  CL.AddDelphiFunction('function EndianInt(i : integer): integer)');
  CL.AddDelphiFunction('function EndianLong(L : longint): longint)');
  CL.AddDelphiFunction('function SwapWord(w : word): word)');
  CL.AddDelphiFunction('function SwapInt(i : integer): integer)');
  CL.AddDelphiFunction('function SwapLong(L : longint): longint)');

  CL.AddDelphiFunction('procedure SymetricCompareFiles(const plaintext, replaintext: string)');
  CL.AddDelphiFunction('procedure PutLinuxLines(const Value: string)');
end;
```

Link to Turbo Power LockBox:

http://lockbox.seanbdurkin.id.au/tiki-index.php

A long poem about cryptography is:

RSA-155

=1094173864157052742180970732204035761200373294544920599091384213147634998428893478471799725789126733249762575289978183379707653724402714674353159335433897

=102639592829741105772054196573991675900716567808038066803341933521790711307779 *
106603488380168454820927220360012878679207958575989291522270608237193062808643

It took 7.4 months (9 weeks, preparation and 5.2 months for the actual factorization) to carry out all the steps.