Personal PNGSafeBox Project

What are we talking about?

Many of us are concerned about the security of our keys, coordinate cards, credit card data, seeds of our bitcoins and any confidential information that we should save and consult from time to time.

We need to consult them often but writing down and taking them with us is not very advisable.

This is the reason to launch this small project for simple data protection.

Idea is the following:

1) Create an image of the document / data to be encrypted.
2) We encrypt it by masking it into a standard graphic format.
3) We will copy - encrypted files - in an App that we can carry in our mobile phone.
4) We can consult it using a key/password decided only by us.
5) We will control both the encryption / decryption application and the Mobile App.

Of course, it is not a 100% guaranteed system but it can give us more reliability than any downloaded application for the following reasons:

* It is based on a simple and personalized obfuscation method just for us.
* We will decide the complexity of the program and develop (or compile) the program.
* We will install the secure files on our mobile (Android) directly.
* We are going to develop our Android Control App.
* We will not need extra permissions for "strange" permissions/access to our terminal.
* In short ... we will control the whole process.

This project is supported in two applications / programs that - united - make up a simple but effective personal protection system for our data.

We can use just one of them, but their joint use allows an absolute control of our data privacy .

The two parts of the project are summarized as follows:

Personal PNGSafeBox Project (I) - Windows Application

An application developed by us - for free - in VB.NET will allow us to obfuscate the graphic files that contain the information to be protected.

This application allows encryption and decryption of such images.

Personal PNGSafeBox Project (II) - Android Application

A Mobile App developed by us - for free - will allow us to transport mentioned images with confidential information in our terminal. This application can query these files (data) securely, decrypting them and show us the information we need on the terminal screen and, of course, will remain encrypted inside our mobile phone.

Even if we lose our mobile phone the files would be difficult to extract as they were natively packed within the application and, in the case of a "rooted" mobile, you would only get obfuscated graphic files that could not be seen unless you know exactly the obfuscation method used.

Part I Personal *PNGSafeBox* for Windows

General description

The project is based on the modification of graphic files (PNG) and whose structure is "tunning" to be practically impregnable.

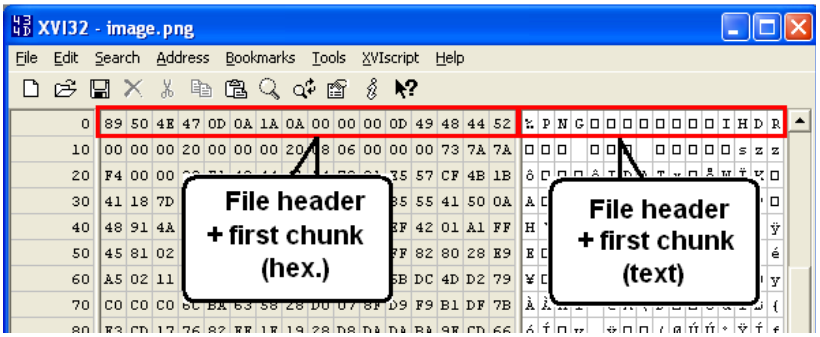
For example, if we want to encrypt an image (photo) that contains the data of our coordinate card, we will only have to convert the image to PNG graphic format and use the program discussed in this article.

The result will be a modified file so that it is not possible to be viewed until it is "reordered" again by the program.

Used encryption algorithm

We will use the typical structure of a PNG graphic file. This can be consulted on the official page of this graphic format (http://www.libpng.org/pub/png/spec/1.2/PNG-Contents.html), but it does not require detailed knowledge since our program is responsible of everything.

A PNG graphic file consists of a header followed by sections that are called "chunks". The specification of each chunk is not the subject of this article, but enought to say that we will use part of the PNG header as well as non-essential part of the first chunk in it.



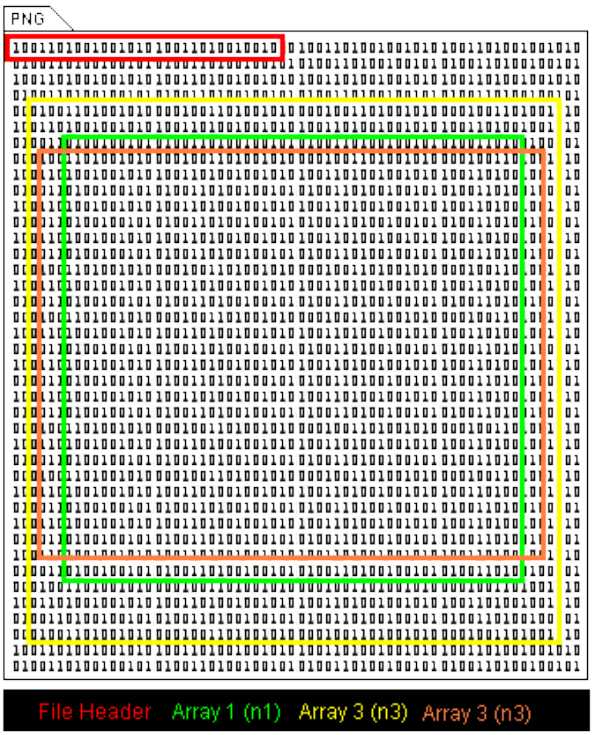
The algorithm to obfuscate the image - in PNG format - to protect it performs these steps:

1) It generates three random numbers of a specific range, N1, N2 and N3
2) Create a byte array with data from offset N1 to EOF-N1 from graphic file.
3) Transpose mentioned bytes and rewrite them back to the file.
4) Repeat twice more - similar procedure for numbers N2 and N3.
5) Create a string with the information structure /N1N2N3+

6) Cipher the string (triple DES) using our KEY and IV.
7) Write the encrypted string in PNG file header position
8) Make some modifications in the initial zone of the header of the file PNG.

Once these steps are done, the PNG file obtained is modified and its visualization is impossible.

The only way to visualize the graphic file again is to use the encryption / decryption program to restore the PNG to its original format.



Software used for the Project

To develop the program we have choose an excellent programming IDE tool that allows us to implement the project in VB.NET completely free of charge.

This is the Sharp Develop environment (http://www.icsharpcode.net/opensource/sd/Default.aspx)

In our case we have used version 3.2.1 software. This is not the last one, but is enough and probably will be much more compatible with Windows systems that are not 100% up-to-date.
The developed program is minimalist and only try to do what it does, that means, encrypt and decrypt PNG files.

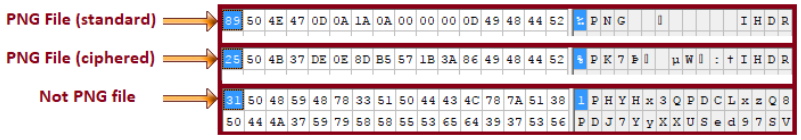
We can download the VB.NET project from the link at the end of this article or from the Webtronika downloads section.

Notes on the program code

Although the program has been developed to be sufficiently self-explanatory (contains many comments) we prefer to explain here some of the its highlights.

The first routine that executes the program is "**Initialize**", which - as its name indicates - will perform certain basic initializations. Once this is done the control is focused on "**Open File**" button, which allows us to choose the file to be operate (encrypt or decrypt).

The action associated with the button will call a check routine - function **IsCiphered ()** - for the selected file. By checking the first byte of this file the program will know if this is a standard PNG file or a file previously encrypted by our program. In case the first byte does not coincide with these two possibilities, file will be qualified as invalid to be processed by our program.



The coding routine - **CodeFile ()** - will execute the obfuscating algorithm discussed in previous lines. It should be noted that the string to be encoded consists of the three positions to be saved (each of them forced to use two characters) and a pair of symbols ("/" and "+") at the beginning and end of the same, that will help in later checks, and of course using Base64 encoding. This causes the final chain to be 8 characters.

For example, if the three randomly obtained positions were 3, 23 and 41 the string **s0** to be encrypted will be **/ 032341 +**

We can see that the decoding routine executes a similar process but when transposing the three file arrays it does - logically - in the opposite order.

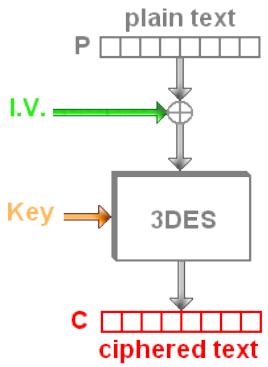
In addition to this, the decryption routine executes the subroutine **ChkDecodeString ()** which controls that the contents of the decrypted string meet the expected format, as already commented for **s0**. For this proposal it makes use of several tests:

* Length of string s0 is 8 characters.
* The characters in the s0 string belong to the base64 numeric base.
* The 6 middle characters are numeric type.
* The start and end characters are / and +.

The routine used to generate the three random numbers has been preset to a range that prevent that generated number(offset to be used later) will be greater than 15, theis way we avoid "touch" the he file header values , because will be where we will later save the encrypted information (using 3DES) for the three random positions.

It is noteworthy that the cyphering of value (string) containing information on the three random positions generated is performed by a triple DES encryption operated in CBC / non-Padding mode. For this we will need to use an encryption KEY (16 bytes) and an initialization vector (8 bytes). Due to the lengths of both strings we preferred to give the user the ability to modify from the front

of the program only the prefixed value of the initialization vector (IV) due this is shorter length and it may be easier for the user to remember (as a "password").



The value of the encryption KEY has been preset in the program (header), but can be modified as neccesary. We only have to respect that the selected string uses only characters belonging to the Base64 encoding.

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

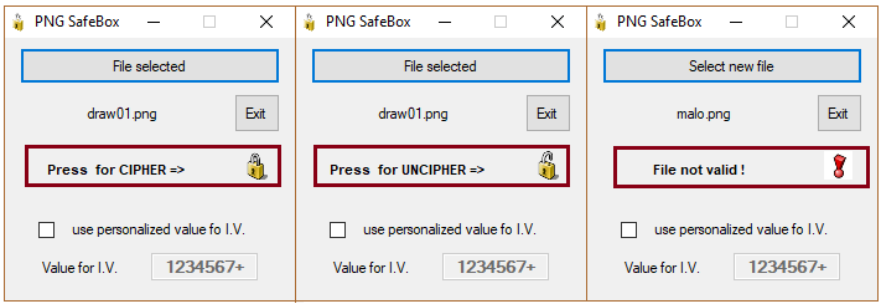
The used transpose routine - **Transpose (F, P)** - is used both in file coding and file decoding. The only difference is that it is called three times and these calls are made in reverse order in the decoding routine.

The **ChkKey ()** routine verifies in real time that the manually selected initialization vector (I.V.) matchs proper length and format. If so, text box IV is green colored, and red if not.

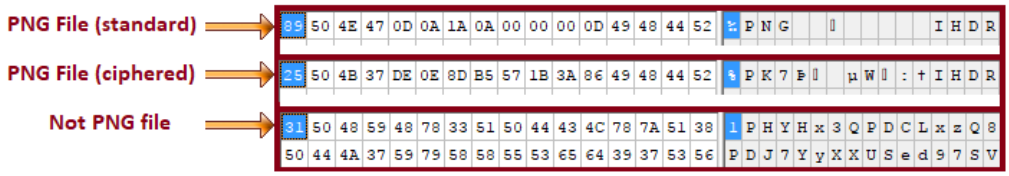
Management of the program

The use of the program has no any secrets. It has been preset to run in a very narrow window and centered on the screen.

To encrypt a file just select the file. Immediately the name of the file will appear on the front of the program and clicking on the padlock icon will encrypt / decrypt the chosen file.



When opening the file, a previous analysis of the file is executed and, as mentioned, depending on the files first byte value the padlock will give us the option to encrypt, decrypt or indicate that the selected file is not valid.



The **byte 0** values that the routine can find are:

- 0x89 => Unencrypted file (standard PNG).
- 0x25 => PNG file encrypted by our program.
- Other => Unrecognized format file.