OpenPuff v4.00 Steganography & Watermarking

Data hiding and watermarking made easy, safe and free

Eng. Cosimo Oliboni, Italy, 2012
Send your suggestions, comments, bug reports, requests
to *oliboni@embeddedsw.net*

OpenPuff HomePage

Legal remarks

Remember: this program was not written for illegal use. Usage of this program that may violate your country's laws is severely forbidden. The author declines all responsibilities for improper use of this program.

No patented code or format has been added to this program.

This program, unlike codecs (encoder/decoder libraries), doesn't process any video or audio data. Ancillary bits only (unused stream bits) are processed. Anything else is simply copied untouched.

This is a Freeware Software

Back

Features: why is this steganography tool different from the others?

OpenPuff is a professional steganography tool, with unique features you won't find among any other free or commercial software. OpenPuff is 100% free and suitable for highly sensitive data covert transmission.
WHAT IS STEGANOGRAPHY?

Let's take a look at its features

- [CARRIERS CHAINS]
  Data is split among many carriers. Only the correct carrier sequence enables unhiding. Moreover, up to 256Mb can be hidden, if you have enough carriers at disposal. Last carrier will be filled with random bits in order to make it undistinguishable from others.

- [SUPPORTED FORMATS]
  Images, audios, videos, flash, adobe.
  SUPPORTED FORMATS IN DETAIL

- [LAYERS OF SECURITY]
  Data, before carrier injection, is encrypted (1), scrambled (2), whitened (3) and encoded (4).
  FEATURES: PROGRAM ARCHITECTURE

  - [LAYER 1 - MODERN MULTI-CRIPTOGRAPHY]
    A set of 16 modern 256bit open-source cryptography algorithms has been joined into a double-password multi-cryptography algorithm (256bit+256bit).

  - [LAYER 2 - CSPRNG BASED SCRAMBLING]
    Encrypted data is always scrambled to break any remaining stream pattern. A new cryptographically secure pseudo random number generator (CSPRNG) is seeded with a third password (256bit) and data is globally shuffled with random indexes.

  - [LAYER 3 - CSPRNG BASED WHITENING]
    Scrambled data is always mixed with a high amount of noise, taken from an independent CSPRNG seeded with hardware entropy.
    OPTIONS: BITS SELECTION LEVEL

  - [LAYER 4 - ADAPTIVE NON-LINEAR ENCODING]
    Whitened data is always encoded using a non-linear function that takes also original carrier bits as input. Modified carriers will need much less change and deceive many steganalysis tests (e.g.: χ^2 test).
    FEATURES: ADAPTIVE ENCODING AND STEGANALYSIS RESISTANCE

  - [EXTRA SECURITY - DENIABLE STEGANOGRAPHY]
    Top secret data can be protected using less secret data as a decoy.
    WHAT IS DENIABLE STEGANOGRAPHY?

- [SOURCE CODE]
  This program relies on the LIBOBFUSCATE system-independent open-source library. Users and developers are absolutely free to link to the core library (100% of the cryptography & obfuscation code), read it and modify it.

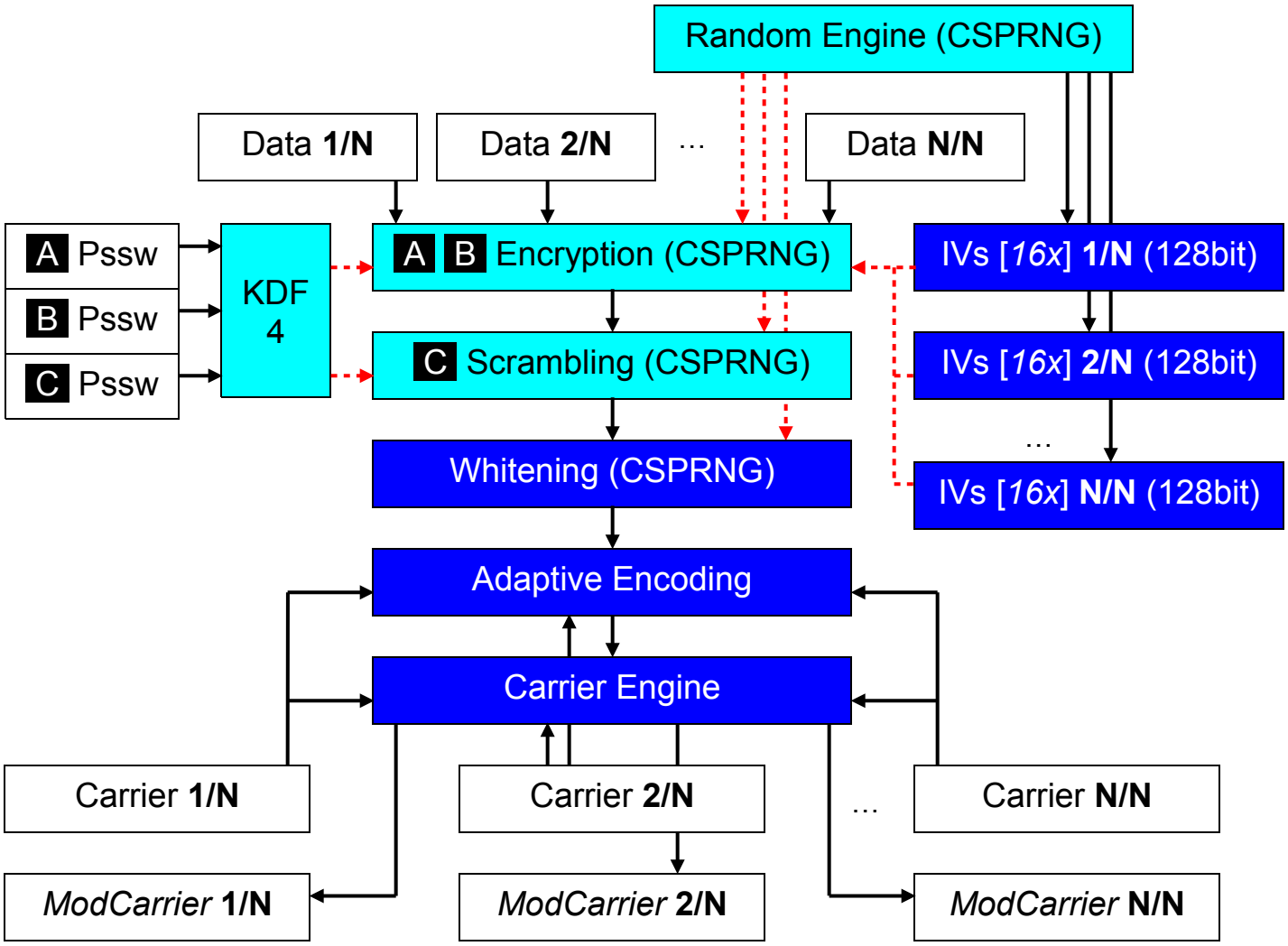
  You're kindly asked to send me any libObfuscate porting/upgrade/customizing/derived sw, in order to analyze them and add them to the project homepage. A central updated official repository will avoid sparseness and unreachability of the project derived code.

BACK

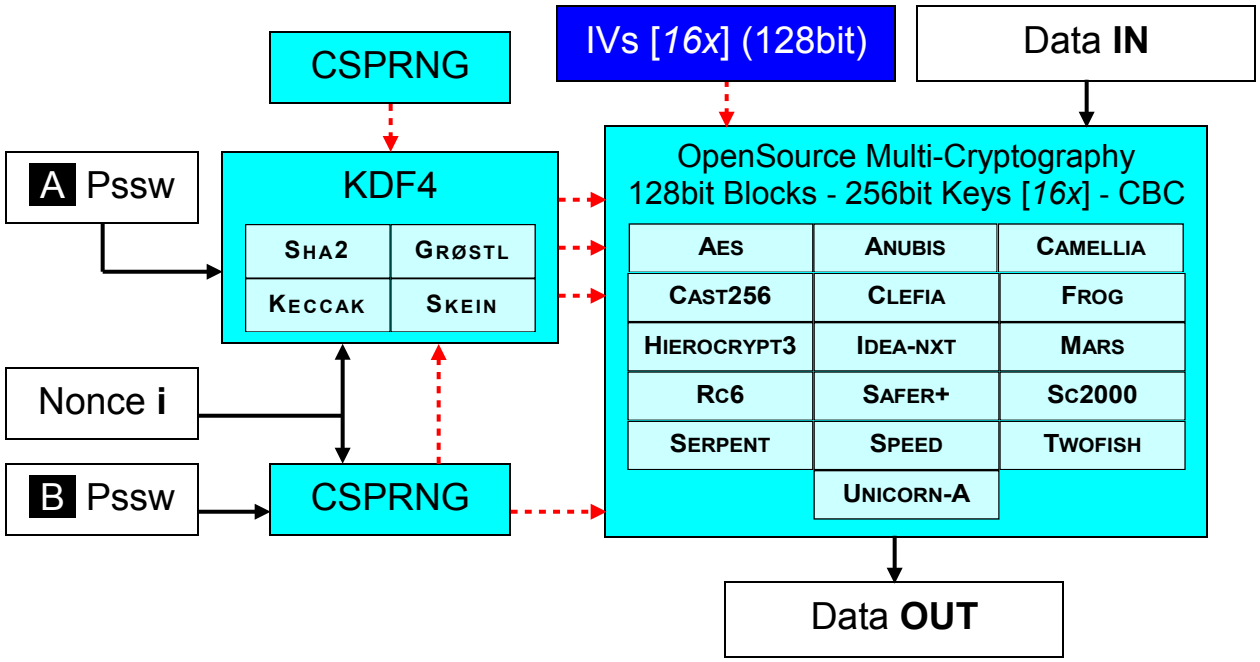
FEATURES: PROGRAM ARCHITECTURE

A high-level global description of OpenPuff's architecture

- data is split among carriers
- each carrier is associated to a random initialization vector array (IVs)
- text passwords (32 characters = 256bit) are associated (KDF4) to hexadecimal passwords
- data is first encrypted with two 256bit KEYS (**A**) (**B**), using multi-cryptography
- encrypted data is then scrambled, with a third key (**C**), to break any remaining stream pattern
- scrambled data is then whitened (= mixed with random noise)
- whitened data is then encoded using a function that takes also original carrier bits as input
- modified carriers receive the processed stream

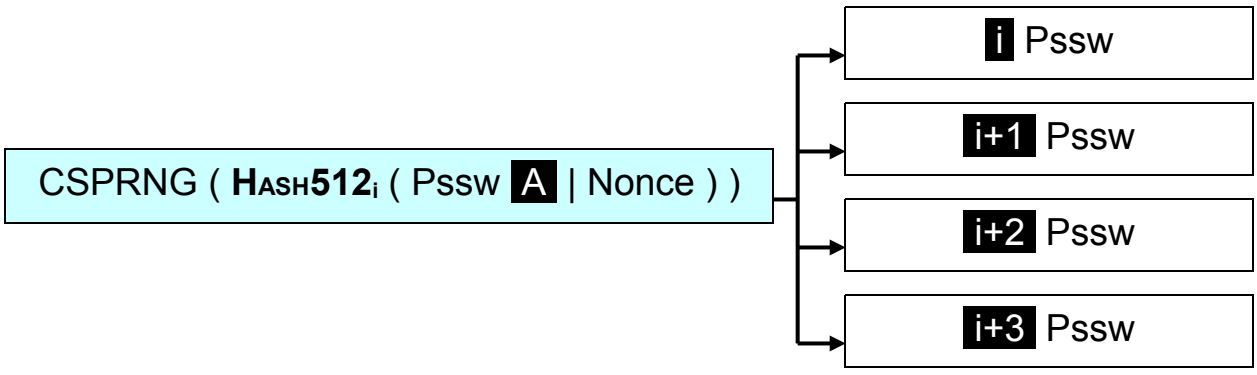


OpenPuff implements multi-cryptography (an advanced kind of PROBABILISTIC ENCRYPTION) joining 16 open-source block-based modern cryptography algorithms, chosen among AES-PROCESS, NESSIE-PROCESS and CRYPTREC-PROCESS. Cypher-Block-Chaining (CBC) wraps these block-based algorithms, letting them to behave as stream-based algorithms.



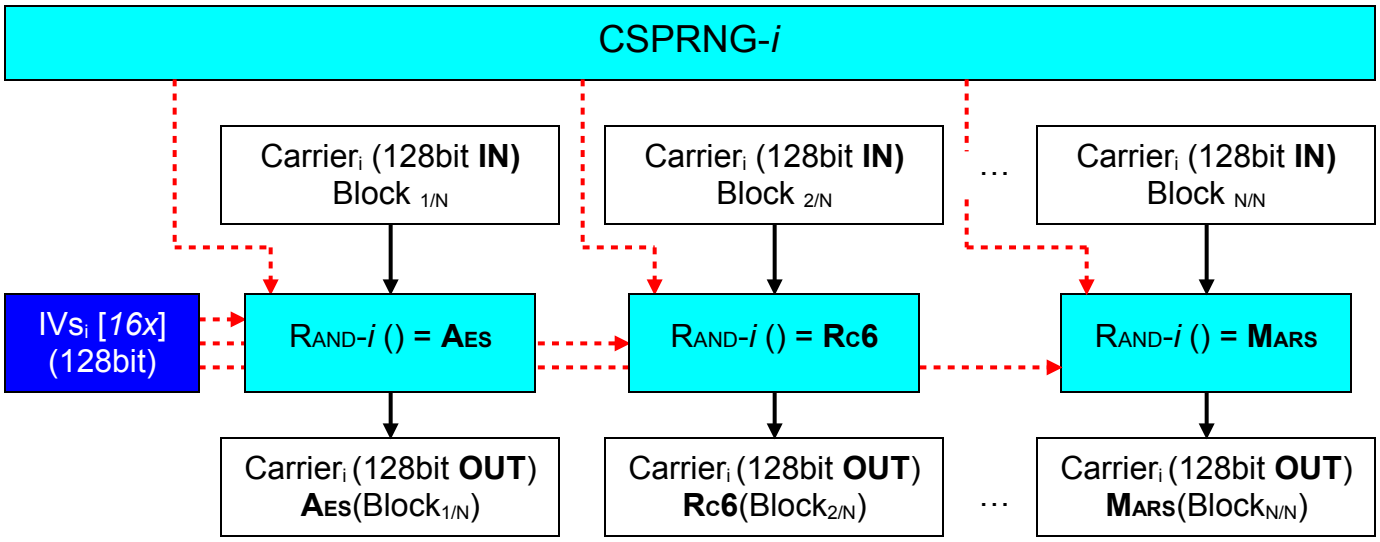
Multi-cryptography setup is a 4 step process

- a random initialization vector array (16 x 128bit) is associated to each carrier
- a pseudo random engine (CSPRNG) is seeded using password (**B**) and a NONCE (the carrier index)
- password (**A**) and the nonce are extended (KDF4) using 4 open-source modern 512bit hashing algorithms, taken from SHA2 and SHA3. Each hash generates four 256bit keys
  Pssw (**1**) | (**2**) | (**3**) | (**4**) = *Rand* (*Sha2* (*Pssw* (**A**) | *Nonce*))
  Pssw (**5**) | (**6**) | (**7**) | (**8**) = *Rand* (*Grøstl* (*Pssw* (**A**) | *Nonce*))
  Pssw (**9**) | (**10**) | (**11**) | (**12**) = *Rand* (*Keccak* (*Pssw* (**A**) | *Nonce*))
  Pssw (**13**) | (**14**) | (**15**) | (**16**) = *Rand* (*Skein* (*Pssw* (**A**) | *Nonce*))
- resulting key array (16 x 256bit) is associated to each cipher using the CSPRNG



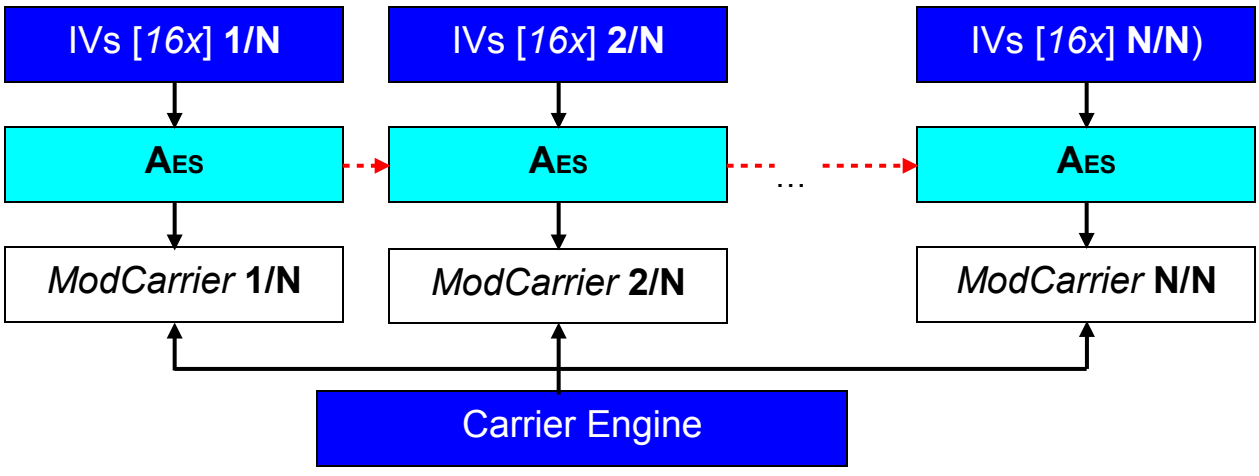
Cryptography is a multi step process

- each carrier gets an independent setup
  $CarrierSetup_i$ = { IVs_i , $CSPRNG_i$, $Keys_i$ }
- each cipher gets an independent setup
  $Cipher_j$ = { IV_j , Key_j }
- each data block is processed with a different cipher, selected using the CSPRNG
  $Carrier_i\ CryptedBlock_k = r \leftarrow Rand\text{-}i\ ()\ ;\ Cipher_r\ (\ IV_r\ ,\ Key_r\ ,\ Carrier_i\ Block_k\ )$

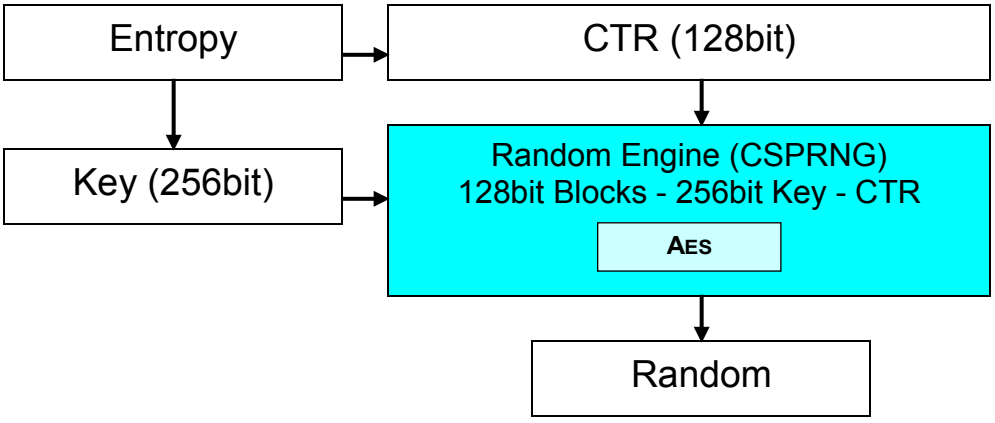


Modified carriers receive

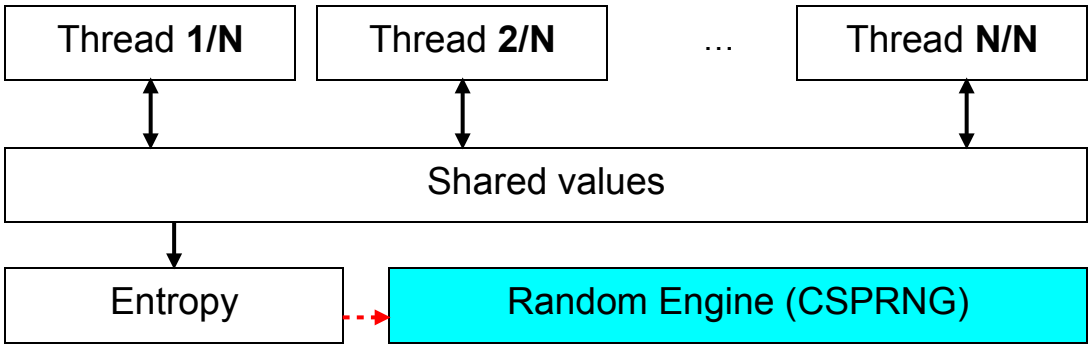
- an encrypted copy of (AES) its initialization vector array
  $CryptedIVs_n = Crypt\ (\ IVs_n\ ,\ CryptedIVs_{n-1}\ )$
- processed data



OpenPuff implements a cryptographically secure pseudo random number generator (CSPRNG) using AES-256 encryption. Block-based secure algorithms running in Counter-Mode (CTR) behave, by construction, as a random engine.



A good hardware source of starting entropy has been provided, not depending on any third-party library or system-API. Threads are always scheduled by the OS in an unpredictable sequence (due to an unavoidable lack of timing accuracy), easily allowing to get a significant amount of EXECUTION RACE CONDITION. ***N*** threads run in parallel, incrementing and decrementing shared values that, after a while, turn into random values.



Testing has been performed on the statistical resistance of the CSPRNG and the multi-wrapper, using the well known PSEUDORANDOM NUMBER SEQUENCE TEST PROGRAM - ENT.

Provided results are taken from 64Kb, 128Kb, ... 256Mb samples:

- bit entropy test resistance:

>7.9999xx / 8.000000	*reference: >7.9*

- compression test resistance (size reduction after compression):

0%	*reference: <1%*

- chi-squared distribution test resistance:

20% < deviazione < 80%	*reference: >10%, <90%*

- mean value test resistance:

127.4x / 127.5	*reference: >127, <128*

- Monte Carlo test resistance:

errore < 0.01%	*reference: < 1%*

- serial correlation test resistance:

< 0.0001	*reference: < 0.01*

BACK

FEATURES: ADAPTIVE ENCODING AND STEGANALYSIS RESISTANCE

Security, performance and steganalysis resistance are conflicting trade-offs.

[Security vs. Performance]: Whitening
- Pro: ensures higher data security
- Pro: allows deniable steganography
- **Con1**: *requires a lot of extra carrier bits*

[Security vs. Steganalysis]: Cryptography + Whitening
- Pro: ensure higher data security
- **Con2**: *their random-like statistical response marks carriers as more "suspicious"*

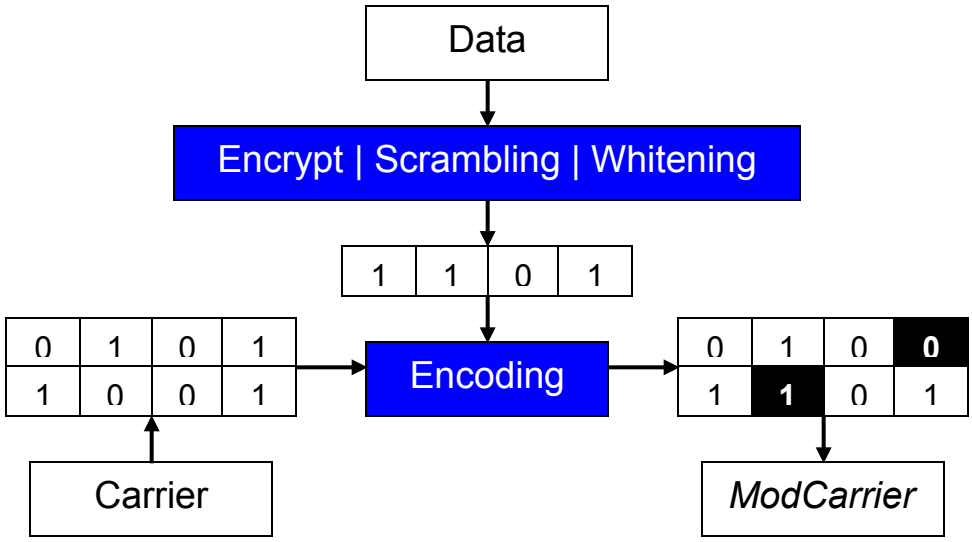
Should we then be concerned about OpenPuff's STEGANALYSIS RESISTANCE? Data, before carrier injection, is encrypted (1), scrambled (2) and whitened (3). Do these 3 steps turn a small amount of hidden data into a big chunk of suspicious data?

A new security layer has been added at the bottom of the data process. Whitened data is always encoded using a non-linear function that takes also original carrier bits as input. Modified carriers will need much less change (**Con1**) and, lowering their random-like statistical response, deceive many steganalysis tests (**Con2**).

"DEFENDING AGAINST STATISTICAL STEGANALYSIS" (Niels Provos)

"CONSTRUCTING GOOD COVERING CODES FOR APPLICATIONS IN STEGANOGRAPHY" (Jessica Fridrich)

The provided coding implementation is a novel unpublished function (built from scratch) that ensures
- output password dependence
- high (50%) embedding efficiency
- low (<20%) change rate



BACK

FAQ 1: Why didn't you simply implement a standard AES-256 or RSA-1024?

Modern open-source cryptography
- has been thoroughly investigated and reviewed by the scientific community
- it's widely accepted as the safest way to secure your data
- fulfills almost every *standard* need of security

OpenPuff doesn't support any CONSPIRACY THEORY against our privacy (SECRET CRACKING BACKDOORS, intentionally weak cryptography designs, …). There's really no reason not to trust standard modern publicly available cryptography (although some old ciphers have been already CRACKED).

Steganography users, however, are very likely to be hiding very sensitive data, with an *unusually high* need of security. Their secrets need to go through a deep process of data OBFUSCATION in order to be able to *longer* survive forensic investigation and hardware aided brute force attacks.

FAQ 2: Is multi-cryptography similar to multiple-encryption?

Multi-cryptography is something really different from MULTIPLE-ENCRYPTION (encrypting more than once). There's really no common agreement about multiple-encryption's reliability. It's thought to be:
- BETTER than single encryption
- WEAK as the weakest cipher in the encryption queue/process
- **worse** than single encryption

OpenPuff supports the last thesis (worse) and never encrypts already encrypted data.

FAQ 3: Is multi-cryptography similar to random/polymorpihc-cryptography?

Random-cryptography, a.k.a. POLYMORPHIC CRYPTOGRAPHY, is a well-known SNAKE-OIL CRYPTOGRAPHY. Multi-cryptography is something completely different and never aims to build some better, random or on-the-fly cipher.

OpenPuff only relies on stable modern opern-source cryptography.

FAQ 4: Is multi-cryptography better than standard cryptography?

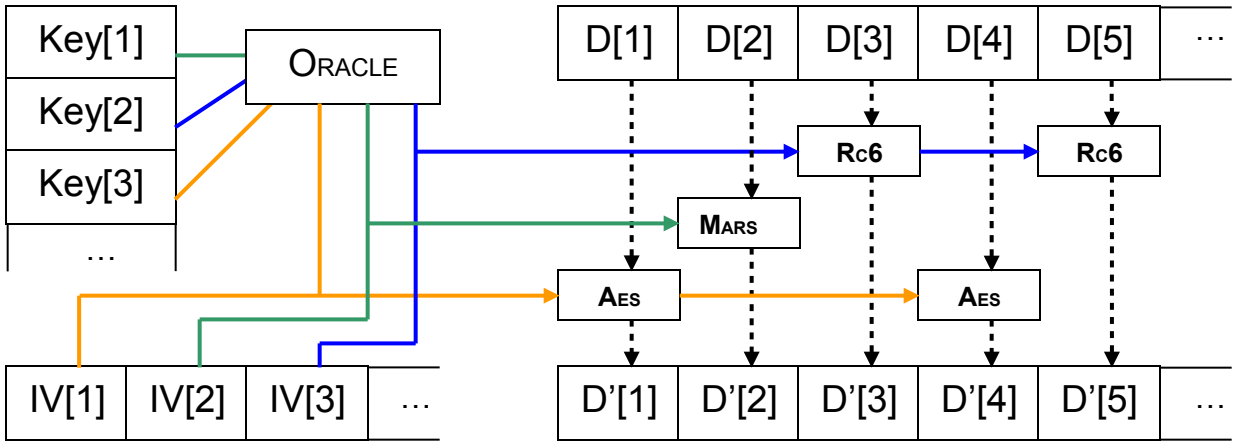
Is a *house* better than a *brick*? No. The house is a *superstructure* and a brick is a *material*.

Is *multi-cryptography* better than *cryptography*? No. Multi-cryptography is part of a data obfuscation *process* and cryptography is a *component*.

Cryptography	=	Brick
Multi-cryptography	=	Floor
Obfuscation process	=	House

Multi-cryptography is a layer of obfuscation

- cryptography setup and CSPRNG setup get two independent passwords
- cryptography setup and CSPRNG setup get also a carrier-order-dependent nonce
- each implemented cipher gets a different IV and key
- CSPRNG behaves like an ORACLE that feeds the cryptography engine during all his choices (which key has to be associated to which cipher, which cipher has to be applied to which data block, …)

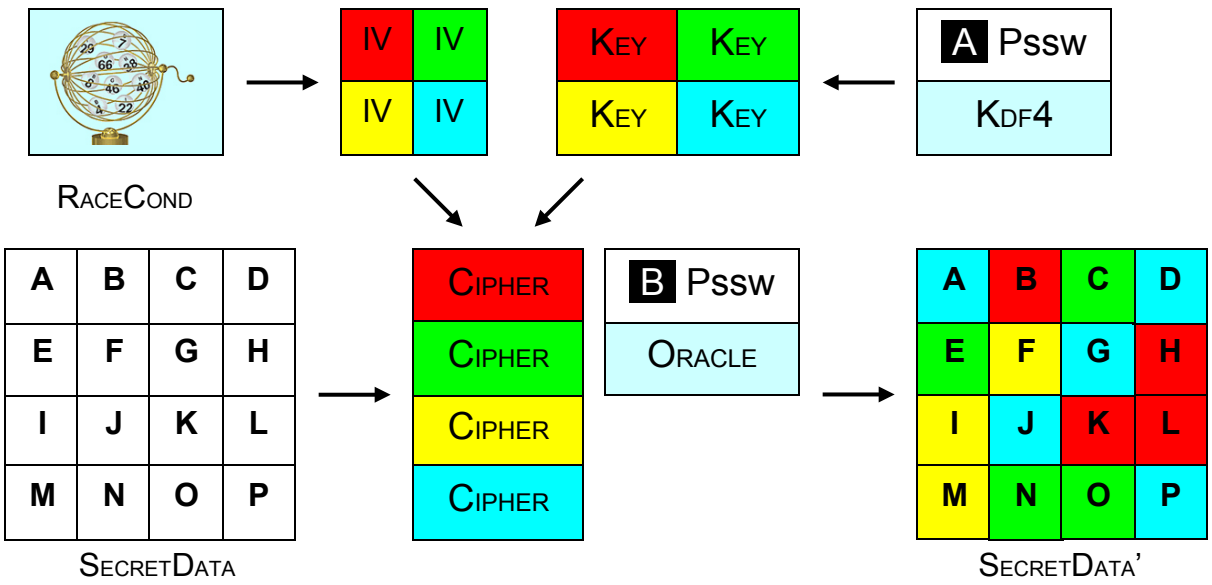


FAQ 5: Is data obfuscation better than standard cryptography?

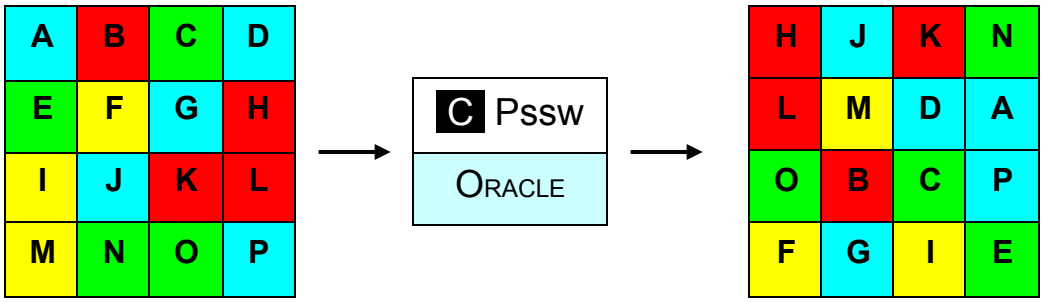
Data obfuscation never raises any claim of UNBREAKABILITY (always to be considered as a symptom of fake software or SNAKE OIL CRYPTOGRAPHY). Yet it's still possible to handle the "*unusually high need of security*"-problem in an effective and constructive way (according to KERCKHOFF'S PRINCIPLE), as an engineering task (*slowing attackers down* as much as possible)

- connecting different obfuscation transformations
- avoiding repeatedly applying the same transformation
- relying only on open-source resources
- applying some global transformation, software-only reversible

[Round 1 – MULTI-CRYPTOGRAPHY (LOCAL TRANSFORMATION)]

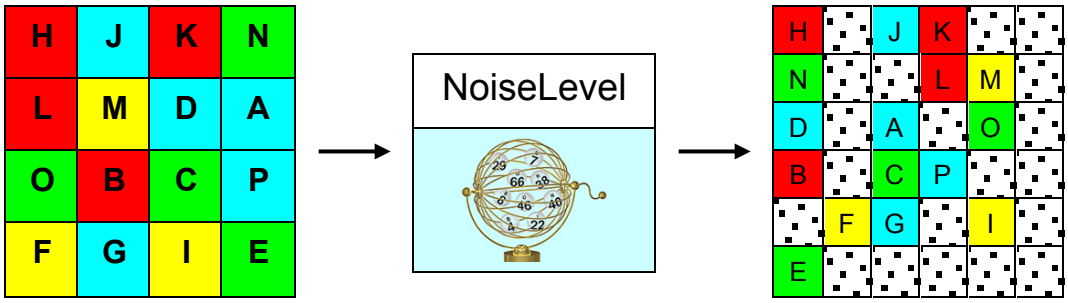


[Round 2 – Scrambling (global transformation)]



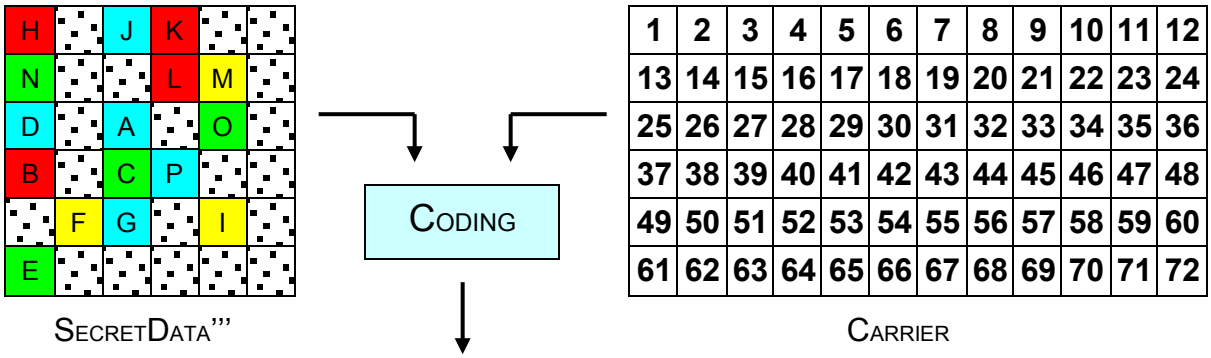
SecretData' SecretData''

[Round 3 – Whitening (global transformation)]

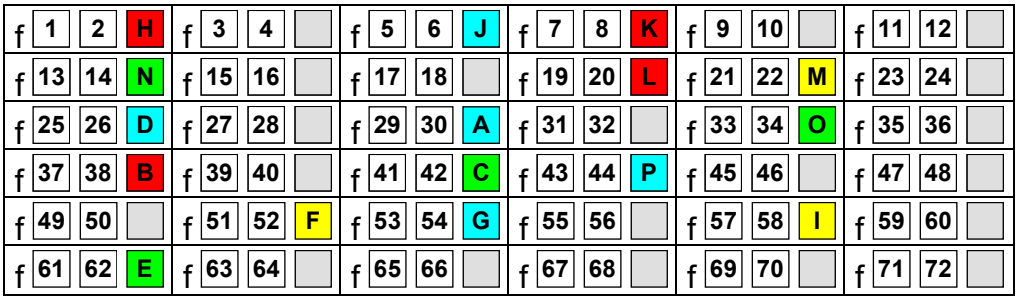


SecretData'' SecretData'''

[Round 4 – Encoding (local transformation)]



SecretData''' Carrier



Carrier'

Features: program architecture

Back

WHAT IS STEGANOGRAPHY?

It's a SMART WAY to hide data into other files, called **carriers**. Modified carriers will look like the original ones, without perceptible changes. Best carriers are videos, images and audio files, since everybody can send/receive/download them without suspects.

The steganography process is highly selective and adaptive, with a minimum payload for each carrier. Carriers with a maximum hidden data amount less than the minimum payload will be discarded.

- +256B → IV
- +16B → a cryptography block

FEATURES: PROGRAM ARCHITECTURE

There's no CARRIER bytes threshold during the marking process.

WHAT IS MARKING?

WHY SHOULD I NEED THIS TECHNIQUE?

You **don't need** this technique if your data

- can be explicitly sent or stored in altered suspicious format.

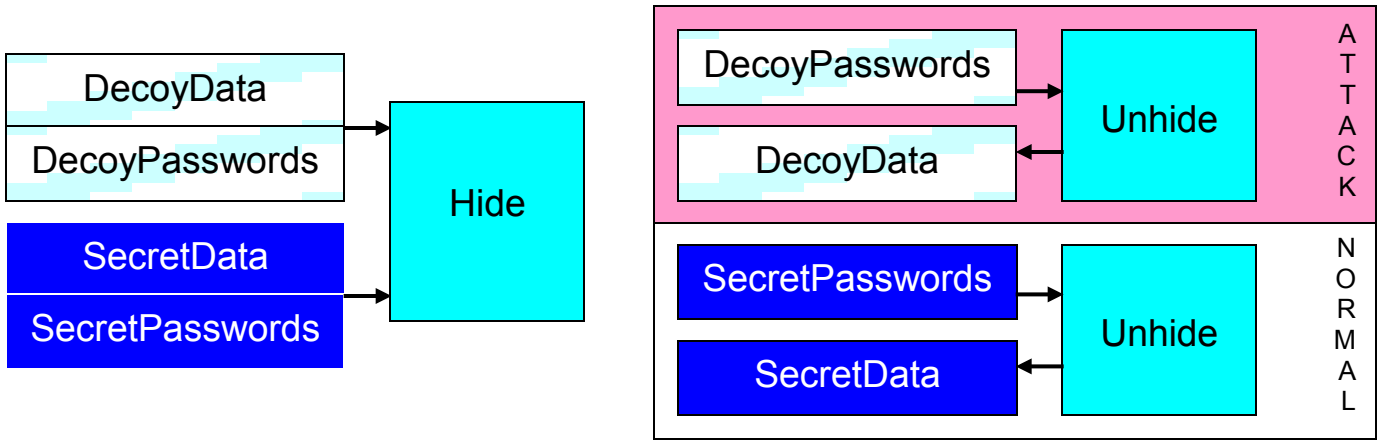
You **may be** interested in this technique if your data

- needs hiding without turning into suspicious format.
- have to be easily accessible by everyone, but retrievable only by people acquainted with your secret intent.

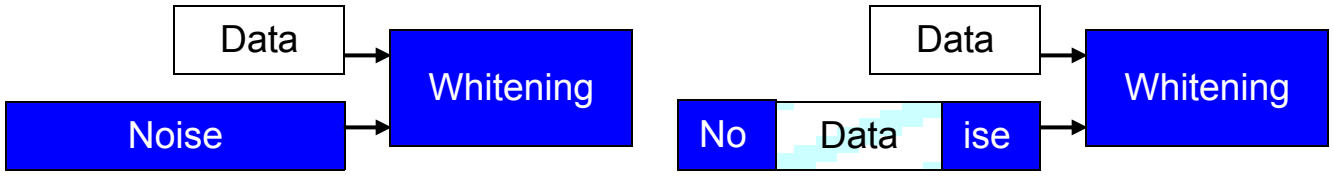
BACK

What is deniable steganography?

Deniable encryption/steganography is a decoy based technique that allows you to convincingly deny the fact that you're hiding **sensitive data**, even if attackers are able to state that you're hiding some data. You only have to provide some expendable decoy data that you would **plausibly** want to keep confidential. It will be revealed to the attacker, claiming that this is all there is.



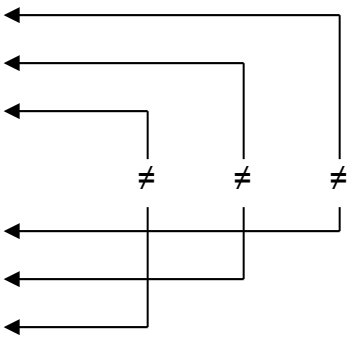
How is it possible? Encrypted and scrambled data, before carrier injection, is whitened (Features: program architecture) with a high amount of noise (Options: bits selection level). Decoy data can replace some of this noise without loosing final properties of Cryptanalysis resistance.



Sensitive data and decoy data are encrypted using different passwords. You have to choose two different sets of different passwords.

Example:

Sensibile data: Password (A) "*FirstDataPssw1*"
Password (B) "*SecondDataPssw2*"
Password (C) "*AnotherDataPssw3*"
(**A** ∩ **B**) 70%, (**A** ∩ **C**) 67%, (**B** ∩ **C**) 68%, Hamming distance ≥ 25%



Decoy data: Password (A') "*FirstDecoyPssw1*"
Password (B') "*SecondDecoyPssw2*"
Password (C') "*AnotherDecoyPssw3*"
(**A'** ∩ **B'**) 72%, (**A'** ∩ **C'**) 60%, (**B'** ∩ **C'**) 70%, Hamming distance ≥ 25%

Each password has to be different (at bit level) and at least 8 characters long.

Example: “DataPssw1” (A) “DataPssw2” (B) “DataPssw3” (C)

(A) 01000100 01100001 01110100 01100001 01010000 01110011 01110011 01110111 00110001
(B) 01000100 01100001 01110100 01100001 01010000 01110011 01110011 01110111 00110010
(C) 01000100 01100001 01110100 01100001 01010000 01110011 01110011 01110111 00110011
(**A** ∩ **B**) 98%, (**A** ∩ **C**) 99%, (**B** ∩ **C**) 99%, HAMMING DISTANCE < 25% **KO**

Example: “FirstDataPssw1” (A) “SecondDataPssw2” (B) “AnotherDataPssw3” (C)

(A) 01000110 01101001 01110010 01110011 01110100 01000100 01100001 01110100 01100001 …
(B) 01010011 01100101 01100011 01101111 01101110 01100100 01000100 01100001 01110100 …
(C) 01000001 01101110 01101111 01110100 01101000 01100101 01110010 01000100 01100001 …
(**A** ∩ **B**) 70%, (**A** ∩ **C**) 67%, (**B** ∩ **C**) 68%, HAMMING DISTANCE ≥ 25% **OK**

You will be asked for

- two **different** sets of different passwords
- a stream of sensitive data
- a stream of decoy data **compatible** (by size) with sensitive data
  $\sum_{k \in \{1, N-1\}} used\_carrier\_bytes(carr_k) < Sizeof(Decoy) \leq \sum_{k \in \{1, N\}} used\_carrier\_bytes(carr_k)$

Example:

Carriers	Carrier bytes	SensitiveData	DecoyData
+Carr (1/N)	32	X	Used
…	2688	X	Used
+Carr (N-1/N)	48	X	Used
+Carr (N/N)	64		Not used
	Total = 2832	Total = 2795	2720 < Size ≤ 2768

BACK

WHAT IS MARKING?

Marking is here stated as the action of signing a file with your copyright mark (best known as WATERMARKING). This program does it in a steganographic way, applied to videos, images and audio files. Your copyright mark will be invisible, but accessible by everyone (using this program), since it won't be password protected.

WHY SHOULD I NEED THIS TECHNIQUE?

You **don't need** this technique if your copyright mark
- needs to be clearly visible
- has to be independent from graphic/audio data, therefore capable of surviving editing operations

You **may be** interested in this technique if your copyright mark
- needs to be invisible
- has to be dependent from graphic/audio data, therefore incapable of surviving editing operations
- has to be accessible by everyone (using this program)

A possible usage of this technique could be: insertion of an invisible copyright mark into registered files that have to be publicly shared. Illegally manipulated copies will maybe look like original ones, but will partially/totally loose the invisible copyright mark.

BACK

SUPPORTED FORMATS IN DETAIL

Images:	BMP, JPG, PCX, PNG, TGA
Audios:	AIFF, MP3, NEXT/SUN, WAV
Videos:	3GP, FLV, MP4, MPG, SWF, VOB
Flash-Adobe:	PDF

Carriers will keep their format

- [in: 32 bits per plane TGA, out: 32 bits per plane TGA]
- [in: Stereo WAV, out: Stereo WAV]
- [in: RGB+Alpha BMP, out: RGB+Alpha BMP]

etc…

Additional tags/chunks and extra bytes will be entirely copied unchanged.
Don't perform any further operation to modified carriers. Their carrier bits would very probably be altered.

BACK

BMP Images (Microsoft)

- Known extensions: **.BMP*, **.DIB*
- 24/32 bits per pixel
- Mono/RGB/RGB+Alpha
- Up to version 5

Back

JPG Images (Joint Photographic Experts Group)

- Known extensions: **.JPG*, **.JPE*, **.JPEG*, **.JFIF*
- 8 bits per plane
- 1-4 planes per pixel, i.e.: Mono/RGB/YCbCr/YCbCrK/CMY/CMYK
- Baseline lossy DCT-jfif with Huffman compression
- h2v2 (4:4), h1v2 (4:2), h2v1 (4:2), h1v1 (4:1) planes independent alignment

Back

PCX Images (ZSoft)

- Known extensions: **.PCX*
- 24 bits per pixel Mono/RGB
- Compressed/Uncompressed

Back

PNG Images (Portable Network Graphics)

- Known extensions: **.PNG*
- 8/16 bits per plan
- Mono/RGB/Mono+Alpha/RGB+Alpha
- Interlaced/Linear

Back

TGA Images (Targa Truevision)

- Known extensions: **.TGA*, **.VDA*, **.ICB*, **.VST*
- Mono-8 bits per pixel or RGB/RGB+Alpha-24/32 bits per pixel
- Compressed/Uncompressed

Back

AIFF Audios (Audio Interchange File Format)

- Known extensions: **.AIF*, **.AIFF*
- 16 bits per sample
- Mono/Stereo/Multi channels
- Linear, uncompressed

Back

MP3 Audios (Fraunhofer Institut)

- Known extensions: **.MP3*
- MPG 1/MPG 2/MPG 2.5 Layer III
- Fixed/Variable Bitrate
- Mono/Dual Channel/Joint Stereo/Stereo
- ID Tagged

Back

NEXT/SUN Audios (Sun & Next)

- Known extensions: **.AU*, **.SND*
- 16 bits per sample
- Mono/Stereo/Multi channels
- Linear, uncompressed

Back

WAV Audios (Microsoft)

- Known extensions: **.WAV, *.WAVE*
- 16 bits per sample
- Mono/Stereo/Multi channels
- PCM, uncompressed

Back

3GP Videos (3rd Generation Partnership Program)

- Known extensions: **.3GP*, **.3GPP*, **.3G2*, **.3GP2*
- Up to version 10
- Codec independent support
- Up to 32 tracks

Back

Adobe FLV Videos (Flash Video)

- Known extensions: **.FLV*, **.F4V*, **.F4P*, **.F4A*, **.F4B*
- Up to version 10
- Codec independent support
- Audio MP3 tracks analysis

Back

MP4 Videos (Motion Picture Experts Group)

- Known extensions: **.MP4*, **.MPG4*, **.MPEG4*, **.M4A*, **.M4V*, **.MP4A*, **.MP4V*
- Up to specification ISO/IEC 14496-12:2008
- Codec independent support
- Up to 32 tracks

Back

MPG Videos (Motion Picture Experts Group)

- Known extensions: **.MPG*, **.MPEG*, **.MPA*, **.MPV*, **.MP1*, **.MPG1*, **.M1A*, **.M1V*, **.MP1A*, **.MP1V*, **.MP2*, **.MPG2*, **.M2A*, **.M2V*, **.MP2A*, **.MP2V*
- Mpeg I Systems - up to specification ISO/IEC 11172-1:1999
- Mpeg II Systems - up to specification ISO/IEC 13818-1:2007
- Codec independent support

Back

Adobe SWF Videos (Shockwave Flash)

- Known extensions: **.SWF*
- Up to version 10
- Codec independent support
- Audio MP3 tracks analysis

Back

VOB Videos (DVD - Video Object)

- Known extensions: **.VOB*
- Mpeg II Systems - up to specification ISO/IEC 13818-1:2007
- Codec independent support

Back

ADOBE PDF FILES (PORTABLE DOCUMENT FORMAT)

- Known extensions: **.PDF*
- Up to specification ISO/IEC 32000-1:2008
- Revision independent support

BACK

SUGGESTIONS FOR BETTER RESULTS

CARRIER CHAINS:

Hide your data into single/multiple carrier chains, adding carriers in unexpected order. Unhiding attempts by unallowed curious people will grow in complexity.

Single carrier example: (Simple, Fast unhiding time, Unsafe)
+MyData >> John.mp3

Single chain example: (Medium complexity, Medium unhiding time, Safe)
+MyData >> Bear.jpg | Zoo.tga | Arrow.png | John.bmp | …

Multiple chains example: (Maximum complexity, Slow unhiding time, Safer)
+MyData (1/n) >> Bear.jpg | Arrow.png | John.bmp | …
…
+MyData (n/n) >> Zoo.tga | Arrow.png | Beep.wav | …

PASSWORD:

Make use of long (>16 chars) easy to remember passwords, changing them every time.

CARRIER BITS SELECTION LEVEL:

Make always use of different levels for each hiding process. Unhiding attempts by unallowed curious people will grow in complexity.

Example:
Hiding process 1:
- **Aiff: Low**
- **BMP: Very low**
- **JPG: Maximum**

…
Hiding process 2:
- **AIFF: Medium**
- **BMP: Low**
- **JPG: Minimum**

…

A FULL SYSTEM COULD BE…

- Hiding your data into many complex chains (hundreds of carriers, with non alphabetical random order), each one with a completely different set of 32-chars-passwords
- Saving all settings inside an “index” single carrier

Example:

+MyData (1/n)	**[carrier1 | … | carrier100]** **[VeryLongPasswords1]** **[BitsSelectionLevel1]**
…	
+MyData (n/n)	**[carrier1 | … | carrier100]** **[VeryLongPasswordsN]** **[BitsSelectionLevelN]**

A very unsuspicious “index” carrier (fixed password + fixed bits selection level) holding a text file that summaries

- carriers name and order
- passwords
- bit selection levels

BACK

Options: bits selection level

(*Minimum*)	1/8 data, 7/8 whitening.
(*Very Low*)	1/7 data, 6/7 whitening.
(*Low*)	1/6 data, 5/6 whitening.
(*Medium*)	1/5 data, 4/5 whitening.
(*High*)	1/4 data, 3/4 whitening.
(*Very High*)	1/3 data, 2/3 whitening.
(*Maximum*)	1/2 data, 1/2 whitening.

Back

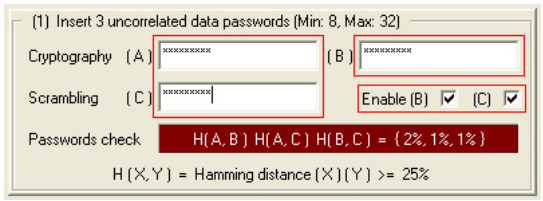
DATA HIDING STEP BY STEP

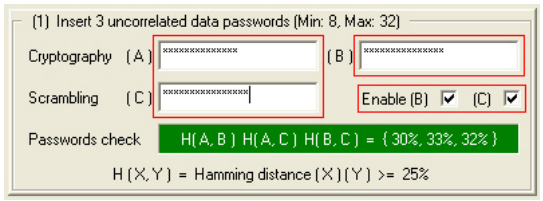
BEGIN:

(*Hide*)	Go to hiding panel

Select *Hide*.

STEP 1:





(*Cryptography A*)	First password (cryptography keys)
(*Cryptography B*)	Second password (cryptography CSPRNG)
(*Scrambling C*)	Third password (scrambling CSPRNG)
(*Enable B*)	Second password enable/disable
(*Enable C*)	Third password enable/disable

Insert three separate passwords. Each password has to be different (at bit level) and at least 8 characters long. Password type and number can be easily customized disabling the second (B) and/or the third (C) password. Disabled passwords will be set as the first (A) password.

Example: “DataPssw1” (A) “DataPssw2” (B) “DataPssw3” (C)

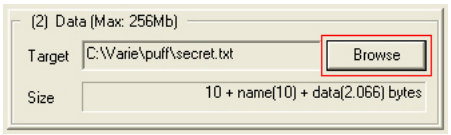
(A) **01000100 01100001 01110100 01100001 01010000 01110011 01110011 01110111 00110001**
(B) **01000100 01100001 01110100 01100001 01010000 01110011 01110011 01110111 00110010**
(C) **01000100 01100001 01110100 01100001 01010000 01110011 01110011 01110111 00110011**
(**A** ∩ **B**) 98%, (**A** ∩ **C**) 99%, (**B** ∩ **C**) 99%, HAMMING DISTANCE < 25% **KO**

Example: “FirstDataPssw1” (A) “SecondDataPssw2” (B) “AnotherDataPssw3” (C)

(A) **01000110 01101001 01110010 01110011 01110100 01000100 01100001 01110100 01100001** …
(B) **01010011 01100101 01100011 01101111 01101110 01100100 01000100 01100001 01110100** …
(C) **01000001 01101110 01101111 01110100 01101000 01100101 01110010 01000100 01100001** …
(**A** ∩ **B**) 70%, (**A** ∩ **C**) 67%, (**B** ∩ **C**) 68%, HAMMING DISTANCE ≥ 25% **OK**

SUGGESTIONS FOR BETTER RESULTS
WHAT IS DENIABLE STEGANOGRAPHY?

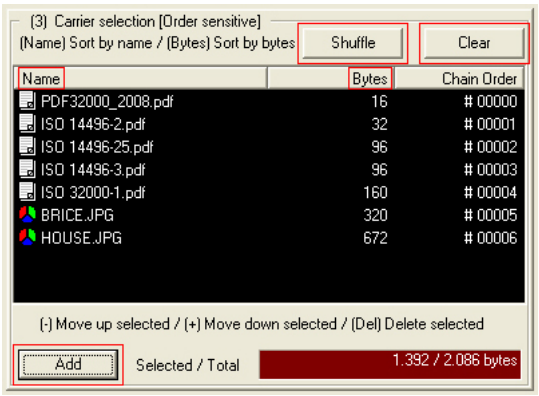
STEP 2:

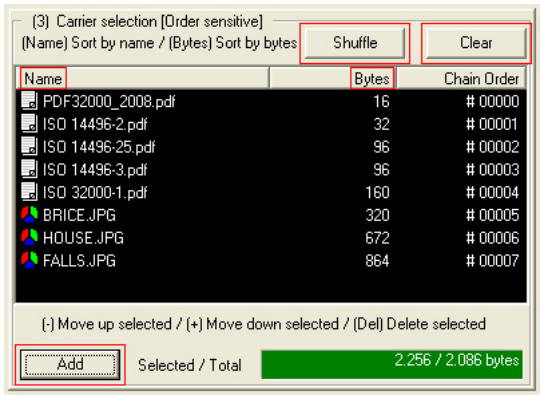


(*Browse*)	Select a file

Choose the secret data you want to hide (typically a zip/rar/… archive).

STEP 3:



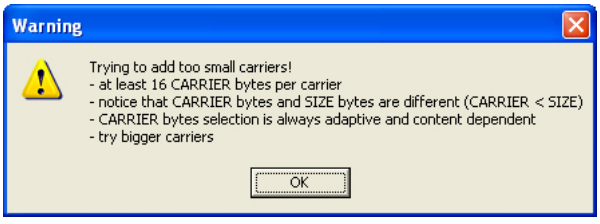


(*Shuffle*)	Random shuffle all carriers
(*Clear*)	Discard all carriers
(*Add*)	Add new carriers to the list
(*Name*)/ (*Bits*)	Sort carriers by name/bits
(*+*)/(*-*)	Move selected carriers up/down
(*Del*)	Delete selected carriers

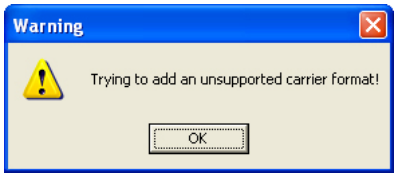
Until *selected bytes* < *total bytes* try

- adding new carriers
- increasing bit selection level

(I)



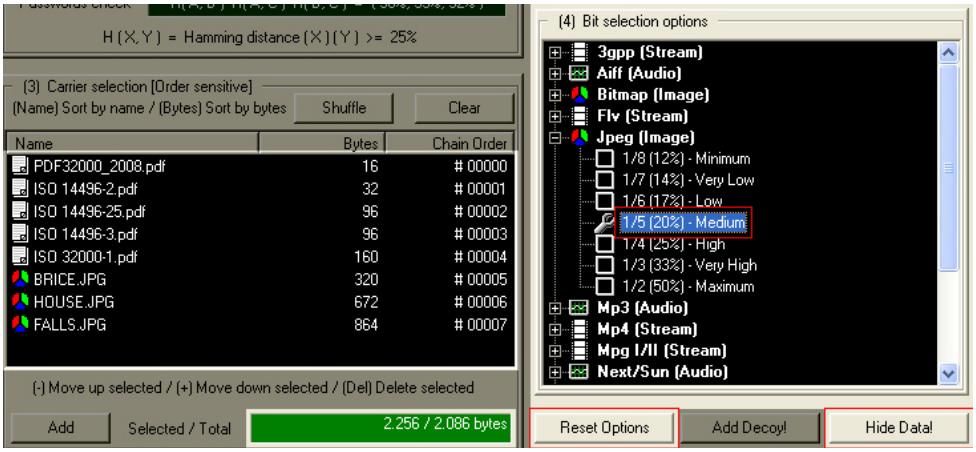
(II)

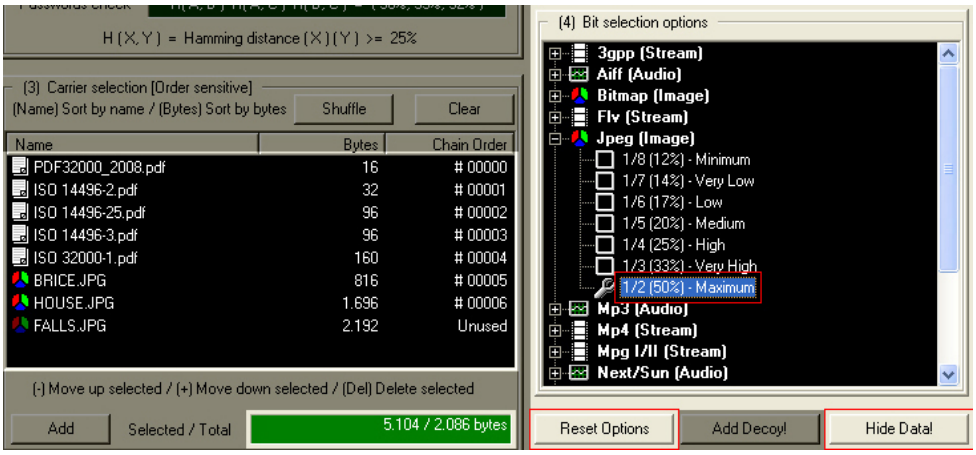


Some carriers will not be added because of steganography-process constraints

- (I) not enough carrier bytes (carrier bytes ⊂ carrier size)
  WHAT IS STEGANOGRAPHY?
- (II) unsupported format
  SUPPORTED FORMATS IN DETAIL

STEP 4:





(*Reset Options*)	Reset all bits selection level to normal
(*Add Decoy!*)	Add a decoy (deniable steganography)
(*Hide!*)	Start hiding

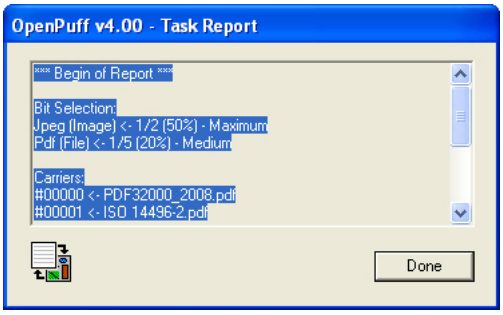
After

- typing twice the same password, at least 8 chars
- selecting a non-empty file to hide
- adding enough carrier bits
- adding a decoy (optional)

start the hiding task

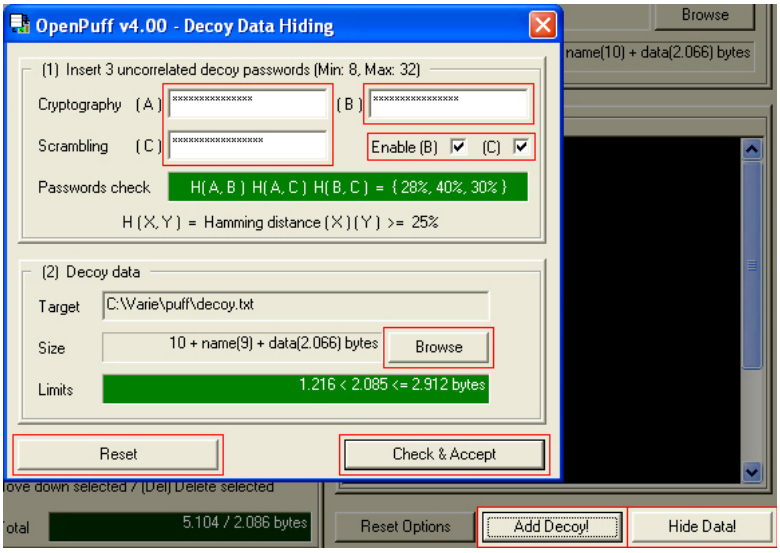
OPTIONS: BITS SELECTION LEVEL

TASK REPORT:



End report summarizes all information needed for a successful unhiding.

STEP 4 – (OPTIONAL):



(*Cryptography A*)	First password (cryptography keys)
(*Cryptography B*)	Second password (cryptography CSPRNG)
(*Scrambling C*)	Third password (scrambling CSPRNG)
(*Enable B*)	Second password enable/disable
(*Enable C*)	Third password enable/disable
(*Browse*)	Select a file
(*Reset*)	Reset password and file
(*Check & Accept*)	Check password correlation and file size

You can also add a decoy password and decoy data

- decoy passwords have to be each other **different**, and different from data passwords
- decoy password type and number can be customized like data passwords
- decoy data has to be **compatible** (by size) with sensitive data
  $\sum_{k \in \{1, N-1\}} used\_carrier\_bytes(carr_k) < Sizeof(Decoy) \leq \sum_{k \in \{1, N\}} used\_carrier\_bytes(carr_k)$

WHAT IS DENIABLE STEGANOGRAPHY?

BACK

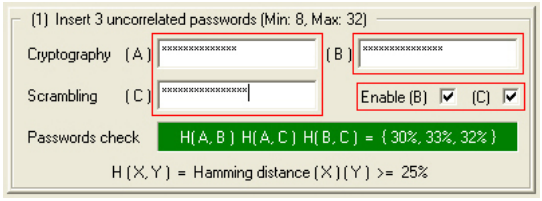
DATA UNHIDING STEP BY STEP

BEGIN:

(*Unhide*)	Go to unhiding panel

Select *Unhide*.

STEP 1:

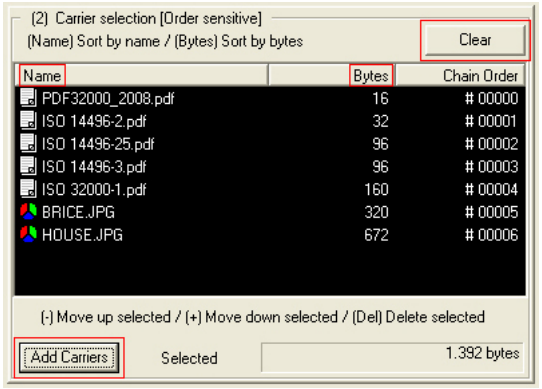


(*Cryptography A*)	First password (cryptography keys)
(*Cryptography B*)	Second password (cryptography CSPRNG)
(*Scrambling C*)	Third password (scrambling CSPRNG)
(*Enable B*)	Second password enable/disable
(*Enable C*)	Third password enable/disable

Insert your passwords (secret to get secret data, decoy to get decoy data), enabling only those used at hiding time.

SUGGESTIONS FOR BETTER RESULTS
WHAT IS DENIABLE STEGANOGRAPHY?

STEP 2:

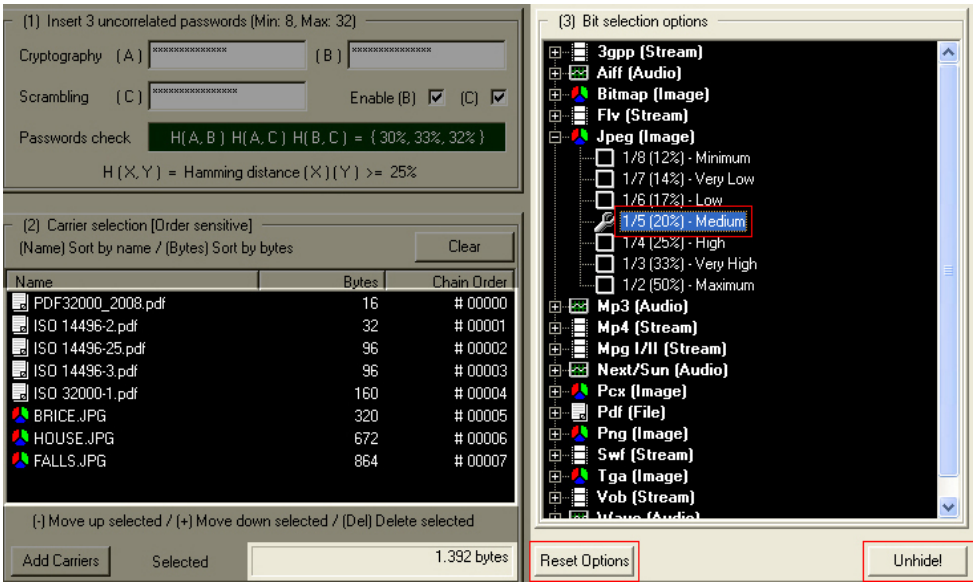


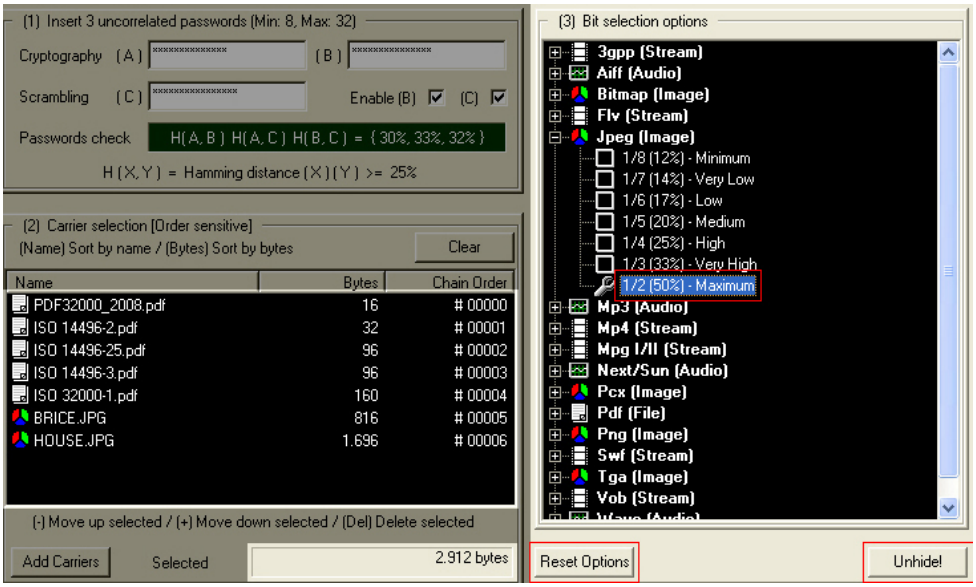
(*Clear*)	Discard all carriers
(*Add*)	Add new carriers to the list
(*Name*)/ (*Bits*)	Sort carriers by name/bits
(*+*)/(*-*)	Move selected carriers up/down
(*Del*)	Delete selected carriers

Add all the carriers that have been processed during the hide task.

SUPPORTED FORMATS IN DETAIL

STEP 3:





(*Reset Options*)	Reset all bits selection level
(*Unhide!*)	Start unhiding

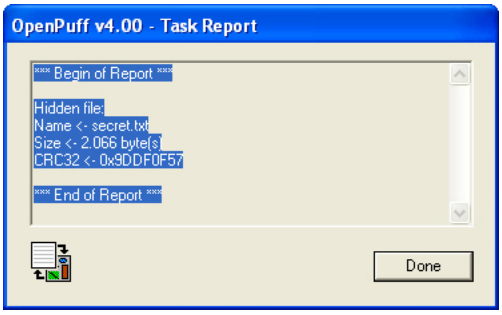
After

- typing twice the same password
- adding all the carriers, in the right order
- setting all bits selection levels to the original value

start the unhiding task

OPTIONS: BITS SELECTION LEVEL

TASK REPORT:



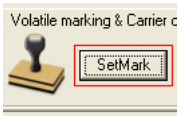
If carriers have been added in the right order, with the original bits selection levels, OpenPuff will be able to reconstruct the original data. For better security, data will be reconstructed only after a successful CRC check.

Even the slightest change in one of the carrier could damage the data and prevent every unhiding try.

BACK

MARK SETTING STEP BY STEP

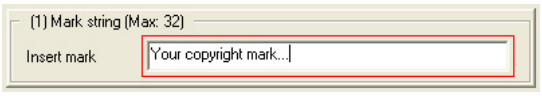
BEGIN:



(*Set Mark*)	Go to mark setting panel

Select *Set Mark*.

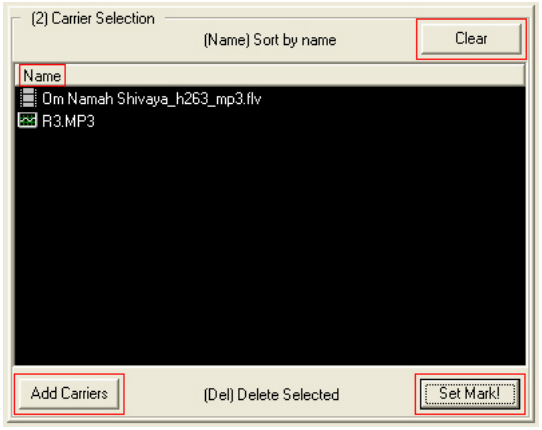
STEP 1:



(*Insert mark*)	Your mark

Type once your mark.

STEP 2:



(*Clear*)	Discard all carriers
(*Add*)	Add new carriers to the list
(*Name*)	Sort carriers by name
(*Del*)	Delete selected carriers
(*Set Mark!*)	Start mark setting

Add all the carriers that need to be marked.
Start the setting task.

SUPPORTED FORMATS IN DETAIL

BACK

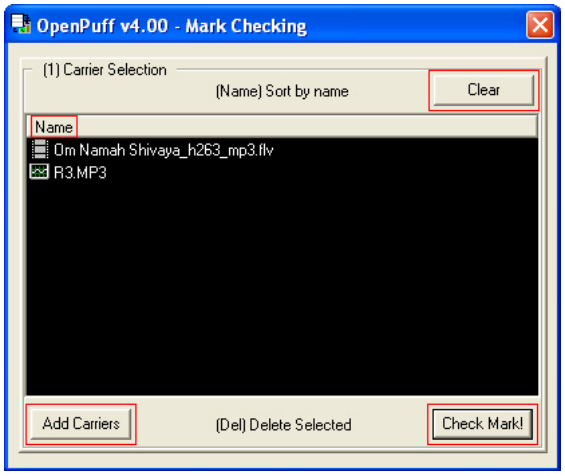
MARK CHECKING STEP BY STEP

BEGIN:

(*Check Mark*)	Go to mark checking panel

Select *Check Mark*.

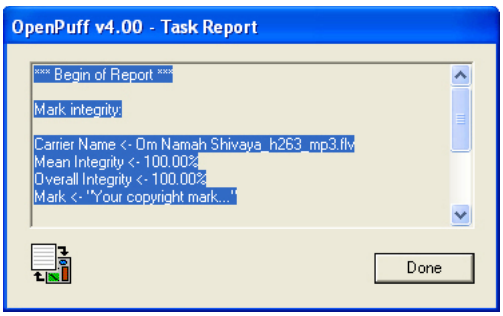
STEP 1:



(*Clear*)	Discard all carriers
(*Add*)	Add new carriers to the list
(*Name*)	Sort carriers by name
(*Del*)	Delete selected carriers
(*Check Mark!*)	Start mark checking

Add all the carriers that need to be checked. Start the checking task.
SUPPORTED FORMATS IN DETAIL

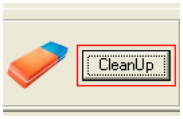
TASK REPORT:



End report summarizes, for each carrier, integrity and mean integrity information.
BACK

DATA & MARK ERASING STEP BY STEP

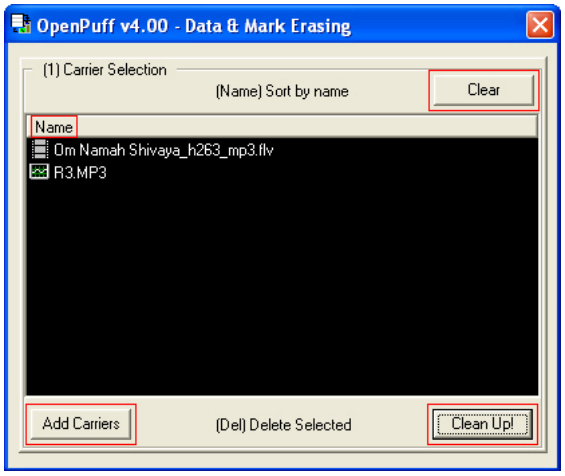
BEGIN:



(*Clean Up*)	Go to data & mark erasing panel

Select *Clean Up*.

STEP 1:



(*Clear*)	Discard all carriers
(*Add*)	Add new carriers to the list
(*Name*)	Sort carriers by name
(*Del*)	Delete selected carriers
(*Clean Up!*)	Start data & mark erasing

Add all the carriers that need to be cleaned and start the cleaning task.

SUPPORTED FORMATS IN DETAIL

BACK