libObfuscate v2.00 reference manual

Advanced file & text locking made easy, safe and free
EmbeddedSW © 2015
Send your suggestions, comments, bug reports, requests
to *embedded@embeddedsw.net*

libObfuscate HomePage

Derived projects: OpenPuff MultiObfuscator

 Legal remarks
 Program architecture

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 is a Free Software

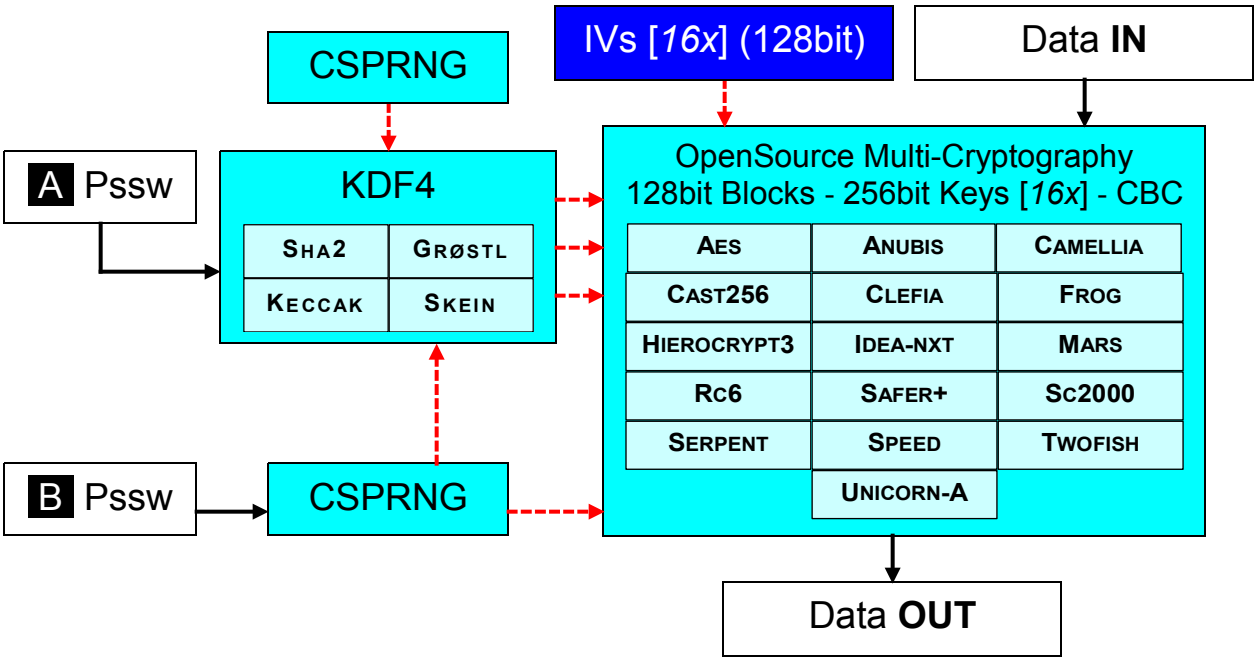
This software is released under GNU LGPL 3.0

You're free to copy, distribute and make commercial use of this software under the following conditions:

- You have to cite the author (and copyright owner): www.embeddedsw.net
- You have to provide a link to the author's Homepage: embeddedsw.net/libobfuscate.html

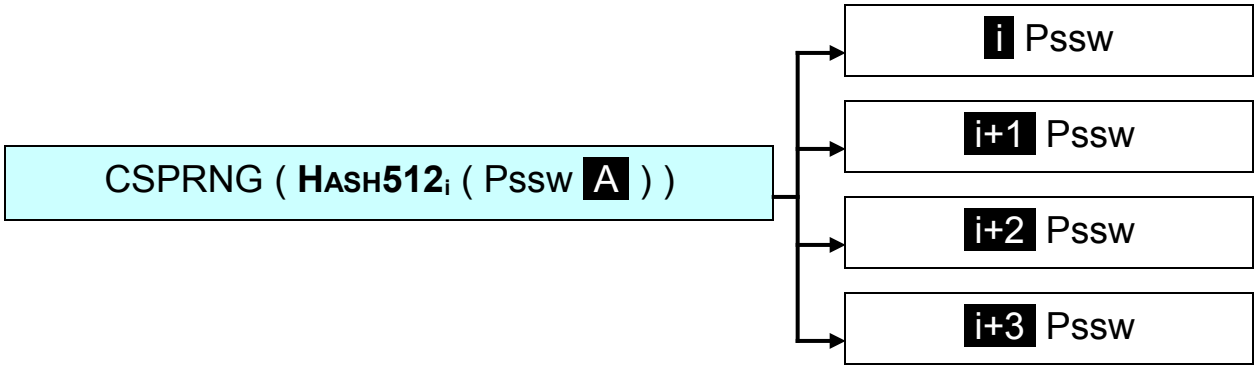
PROGRAM ARCHITECTURE

libObfuscate 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.



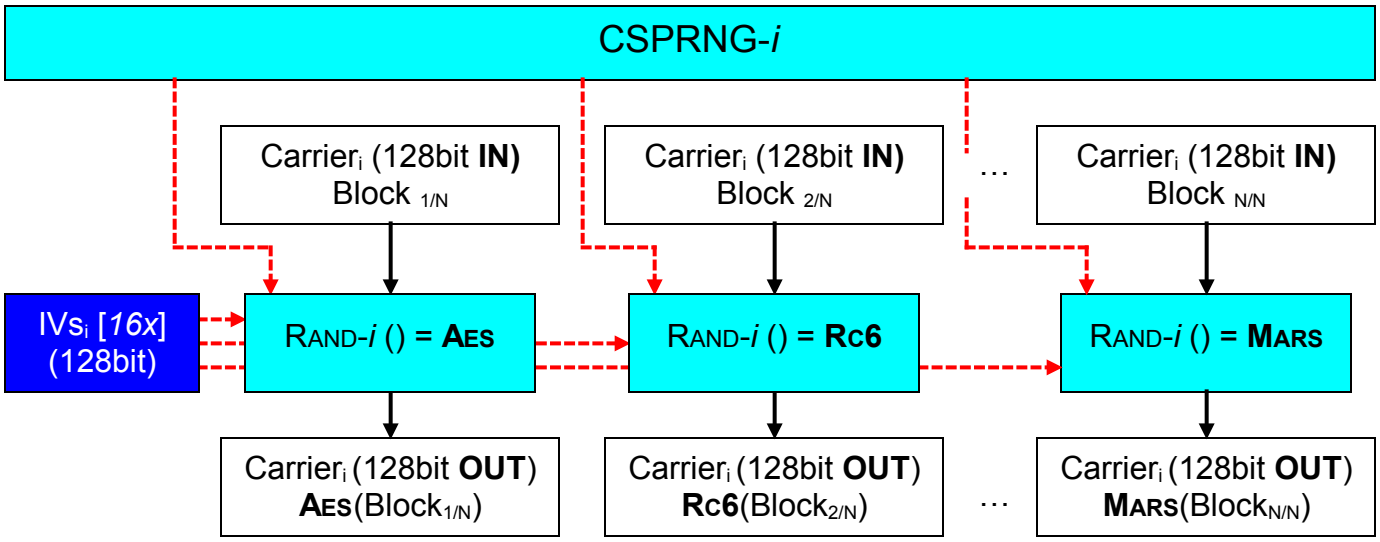
Multi-cryptography setup is a 4 step process

- a random initialization vector array (16 x 128bit) is associated to each carrier
- a pseudo random engine (CSPRNG) is seeded using password (**B**)
- password (**A**) is 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**)))
  Pssw (**5**) | (**6**) | (**7**) | (**8**) = *Rand* (*Grøstl* (*Pssw* (**A**)))
  Pssw (**9**) | (**10**) | (**11**) | (**12**) = *Rand* (*Keccak* (*Pssw* (**A**)))
  Pssw (**13**) | (**14**) | (**15**) | (**16**) = *Rand* (*Skein* (*Pssw* (**A**)))
- resulting key array (16 x 256bit) is associated to each cipher using the CSPRNG



Cryptography is a multi step process

- each data gets a global setup
  Setup = { { *IV* } , *CSPRNG* , { *Key* } }
- 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
  $CryptedBlock_k$ = *r* ← *Rand-i* () ; $Cipher_r$ (IV_r , Key_r , $Block_k$)



- cryptography setup and CSPRNG setup get two independent passwords
- 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, …)