PyCryptodome Documentation

Release 3.3

Legrandin

October 28, 2015

Contents

CHAPTER 1

Introduction

PyCryptodome is a self-contained, public domain Python package of low-level cryptographic primitives.

It supports Python 2.4 or newer, all Python 3 versions and PyPy.

All the code can be downloaded from GitHub.

PyCryptodome is not a wrapper to a separate C library like *OpenSSL*. To the largest possible extent, algorithms are implemented in pure Python. Only the pieces that are extremely critical to performance (e.g. block ciphers) are implemented as C extensions.

1.1 News

- **29 Oct 2015 (NEW)**. Release 3.3.
- 9 Sep 2015. Minor release 3.2.1.
- 6 Sep 2015. Release 3.2.
- 15 Mar 2015. Release 3.1.
- 24 Jun 2014. Release 3.0.

1.2 PyCryptodome and PyCrypto

PyCryptodome is a fork of the PyCrypto project.

It brings the following enhancements with respect to the last official version of PyCrypto (2.6.1):

- Authenticated encryption modes (GCM, CCM, EAX, SIV, OCB)
- Accelerated AES on Intel platforms via AES-NI
- First class support for PyPy
- SHA-3 (including SHAKE XOFs) and BLAKE2 hash algorithms
- Salsa20 stream cipher
- scrypt and HKDF
- Deterministic DSA
- Password-protected PKCS#8 key containers

- Shamir's Secret Sharing scheme
- Random numbers get sourced directly from the OS (and not from a CSPRNG in userspace)
- Simplified install process, including better support for Windows
- FIPS 186-4 compliant RSA key generation
- Major clean ups and simplification of the code base

The fork took place because of the very bad state PyCrypto was in, and the little maintanance it was receiving.

CHAPTER 2

Features

This page lists the low-level primitives that PyCryptodome provides.

You are expected to have a solid understanding of cryptography and security engineering to successfully use them.

You must also be able to recognize that some primitives are obsolete (e.g. TDES) or even unsecure (RC4). They are provided only to enable backward compatibility where required by the applications.

A list of useful resources in that area can be found on Matthew Green's blog.

- Symmetric ciphers:
  - AES
  - Single and Triple DES
  - CAST-128
  - RC2
- Traditional modes of operations for symmetric ciphers:
  - ECB
  - CBC
  - CFB
  - OFB
  - CTR
  - OpenPGP (a variant of CFB, RFC4880)
- AEAD modes of operations for symmetric ciphers:
  - CCM (AES only)
  - EAX
  - GCM (AES only)
  - SIV (AES only)
  - OCB (AES only)
- Stream ciphers:
  - Salsa20
  - ChaCha20
  - RC4

- Cryptographic hashes:
  - SHA-1
  - SHA-2 family (224, 256, 384, 512)
  - SHA-3 family (224, 256, 384, 512)
  - BLAKE2b and BLAKE2s
  - RIPE-MD160
  - MD5
- Message Authentication Codes (MAC):
  - HMAC
  - CMAC
- Asymmetric key generation:
  - RSA
  - DSA
  - ElGamal
- Export and import format for asymmetric keys:
  - PEM (clear and encrypted)
  - PKCS#8 (clear and encrypted)
  - ASN.1 DER
- Asymmetric ciphers:
  - PKCS#1 (RSA)
    - RSAES-PKCS1-v1_5
    - RSAES-OAEP
- Asymmtric digital signatures:
  - PKCS#1 (RSA)
    - RSASSA-PKCS1-v1_5
    - RSASSA-PSS
  - DSA
    - FIPS 186-3
    - Deterministic (RFC6979)
- Key derivation:
  - PBKDF1
  - PBKDF2
  - scrypt
  - HKDF
- Other cryptographic protocols:
  - Shamir Secret Sharing

- Padding
  - PKCS#7
  - ISO-7816
  - X.923

CHAPTER 3

Installation

The procedures below all perform the same actions:

1. Install `virtualenv` and `pip`
2. Create a virtual environment
3. Download PyCryptodome from pypi
4. *(In Unix only)* Compile the C extensions of PyCryptodome
5. Install PyCryptodome in the virtual environment
6. Run the test suite to verify that all algorithms work correctly

Note: PyCryptodome resides in the same namespace of PyCrypto (`Crypto`). In order to avoid any possible conflict, these instructions do not install PyCryptodome at the system level.

3.1 Linux Ubuntu

For Python 2.x:

```
$ sudo apt-get install build-essential libgmp3c2
$ sudo apt-get install python-virtualenv python-dev
$ virtualenv -p /usr/bin/python2 MyProject
$ cd MyProject
$ . bin/activate
$ pip install pycryptodome
$ python -m Crypto.SelfTest
```

For Python 3.x:

```
$ sudo apt-get install build-essential libgmp3c2
$ sudo apt-get install python-virtualenv python3-dev
$ virtualenv -p /usr/bin/python3 MyProject
$ cd MyProject
$ . bin/activate
$ pip install pycryptodome
$ python3 -m Crypto.SelfTest
```

For PyPy:

```
$ sudo apt-get install build-essential libgmp3c2
$ sudo apt-get install python-virtualenv pypy-dev
$ virtualenv -p /usr/bin/pypy MyProject
$ cd MyProject
$ . bin/activate
$ pip install pycryptodome
$ pypy -m Crypto.SelfTest
```

3.2 Linux Fedora

For Python 2.x:

```
$ sudo yum install gcc gmp
$ sudo yum install python-virtualenv python-devel
$ virtualenv -p /usr/bin/python2 MyProject
$ cd MyProject
$ . bin/activate
$ pip install pycryptodome
$ python -m Crypto.SelfTest
```

For Python 3.x:

```
$ sudo yum install gcc gmp
$ sudo yum install python3-virtualenv python3-devel
$ virtualenv -p /usr/bin/python3 MyProject
$ cd MyProject
$ . bin/activate
$ pip install pycryptodome
$ python3 -m Crypto.SelfTest
```

For PyPy:

```
$ sudo yum install gcc gmp
$ sudo yum install python-virtualenv pypy-devel
$ virtualenv -p /usr/bin/pypy MyProject
$ cd MyProject
$ . bin/activate
$ pip install pycryptodome
$ pypy -m Crypto.SelfTest
```

3.3 Windows (pre-compiled)

1. Make sure that the `PATH` environment variable contains the directory of your Python interpreter and its subdirectory `Scripts`.

   Typically, that means typing something like this at the command prompt:

   ```
   > set PATH=%PATH%;C:\Python27;C:\Python27\Scripts
   ```

   or:

   ```
   > set PATH=%PATH%;C:\Python34;C:\Python34\Scripts
   ```

2. **[Only once. Skip if you have Python 3.4 or newer]** Install `pip` by downloading and executing the Python script get-pip.py:

```
> python get-pip.py
```

3. **[Only once]** Install `virtualenv` with:

```
> pip install virtualenv
```

4. Create a virtual environment for your project:

```
> cd %USERPROFILE%
> virtualenv MyProject
> cd MyProject
> Scripts\activate
```

5. Install PyCryptodome as a wheel:

```
> pip install pycryptodome
```

6. To make sure everything works fine, run the test suite:

```
> python -m Crypto.SelfTest
```

3.4 Windows (from sources, Python 2.x, Python <=3.2)

Windows does not come with a C compiler like most Unix systems. The simplest way to compile the *Pycryptodome* extensions from source code is to install the minimum set of Visual Studio components freely made available by Microsoft.

1. Ensure you have *pip* and *virtualenv* installed (see previous section).
2. Run Python from the command line and note down its version and whether it is a 32 bit or a 64 bit application.

   For instance, if you see:

```
Python 2.7.2+ ... [MSC v.1500 32 bit (Intel)] on win32
```

   you clearly have Python 2.7 and it is a 32 bit application.

3. **[Only once]** In order to speed up asymmetric key algorithms like RSA, it is recommended to install the MPIR library (a fork of the popular GMP library, more suitable for the Windows environment). For convenience, I made available pre-compiled *mpir.dll* files to match the various types of Python one may have:

   - Python 2.x, 3.1, 3.2 (VS2008 runtime)
     - 32 bits
     - 64 bits
   - Python 3.3 and 3.4 (VS2010 runtime)
     - 32 bits
     - 64 bits
   - Python 3.5 (VS2015 runtime)
     - 32 bits
     - 64 bits

   Download the correct *mpir.dll* and drop it into the Python interpreter directory (for instance `C:\Python34`). *Pycryptodome* will automatically make use of it.

4. **[Only once]** Install Virtual Clone Drive.
5. **[Only once]** Download the ISO image of the **'MS SDK for Windows 7 and . NET Framework 3.5 SP1'_**. It contains the Visual C++ 2008 compiler.

   There are three ISO images available: you will need `GRMSDK_EN_DVD.iso` if your Windows OS is 32 bits or `GRMSDKX_EN_DVD.iso` if 64 bits.

   Mount the ISO with *Virtual Clone Drive* and install the C/C++ compilers and the redistributable only.
6. If your Python is a 64 bit application, open a command prompt and perform the following steps:

```
> cd "C:\Program Files\Microsoft SDKs\Windows\v7.0"
> cmd /V:ON /K Bin\SetEnv.Cmd /x64 /release
> set DISTUTILS_USE_SDK=1
```

   Replace `/x64` with `/x86` if your Python is a 32 bit application.
7. Enter the virtual environment for your project:

```
> cd %USERPROFILE%
> cd MyProject
> Scripts\activate
```

8. Compile and install PyCryptodome:

```
> pip install pycryptodome --no-use-wheel
```

9. To make sure everything work fine, run the test suite:

```
> python -m Crypto.SelfTest
```

3.5 Windows (from sources, Python 3.3 and 3.4)

Windows does not come with a C compiler like most Unix systems. The simplest way to compile the *Pycryptodome* extensions from source code is to install the minimum set of Visual Studio components freely made available by Microsoft.

1. Ensure you have *pip* and *virtualenv* installed (see previous section).
2. Run Python from the command line and note down its version and whether it is a 32 bit or a 64 bit application.

   For instance, if you see:

```
Python 2.7.2+ ... [MSC v.1500 32 bit (Intel)] on win32
```

   you clearly have Python 2.7 and it is a 32 bit application.
3. **[Only once]** In order to speed up asymmetric key algorithms like RSA, it is recommended to install the MPIR library (a fork of the popular GMP library, more suitable for the Windows environment). For convenience, I made available pre-compiled *mpir.dll* files to match the various types of Python one may have:
   - Python 2.x, 3.1, 3.2 (VS2008 runtime)
     - 32 bits
     - 64 bits
   - Python 3.3 and 3.4 (VS2010 runtime)
     - 32 bits
     - 64 bits

- Python 3.5 (VS2015 runtime)
  - 32 bits
  - 64 bits

Download the correct *mpir.dll* and drop it into the Python interpreter directory (for instance `C:\Python34`). *Pycryptodome* will automatically make use of it.

4. **[Only once]** Install Virtual Clone Drive.
5. **[Only once]** Download the ISO image of the **'MS SDK for Windows 7 and . NET Framework 4'_**. It contains the Visual C++ 2010 compiler.

   There are three ISO images available: you will need `GRMSDK_EN_DVD.iso` if your Windows OS is 32 bits or `GRMSDKX_EN_DVD.iso` if 64 bits.

   Mount the ISO with *Virtual Clone Drive* and install the C/C++ compilers and the redistributable only.
6. If your Python is a 64 bit application, open a command prompt and perform the following steps:

```
> cd "C:\Program Files\Microsoft SDKs\Windows\v7.1"
> cmd /V:ON /K Bin\SetEnv.Cmd /x64 /release
> set DISTUTILS_USE_SDK=1
```

   Replace `/x64` with `/x86` if your Python is a 32 bit application.
7. Enter the virtual environment for your project:

```
> cd %USERPROFILE%
> cd MyProject
> Scripts\activate
```

8. Compile and install PyCryptodome:

```
> pip install pycryptodome --no-use-wheel
```

9. To make sure everything work fine, run the test suite:

```
> python -m Crypto.SelfTest
```

3.6 Windows (from sources, Python 3.5 and newer)

Windows does not come with a C compiler like most Unix systems. The simplest way to compile the *Pycryptodome* extensions from source code is to install the minimum set of Visual Studio components freely made available by Microsoft.

1. **[Once only]** Download MS Visual Studio 2015 (Community Edition) and install the C/C++ compilers and the redistributable only.
2. Perform all steps from the section *Windows (pre-compiled)* but add the `--no-use-wheel` parameter when calling `pip`:

```
> pip install pycryptodome --no-use-wheel
```

CHAPTER 4

API documentation

The API can be found here. Soon it will be moved on these pages.

CHAPTER 5

Examples

5.1 Encrypt data with AES

The following code generates a new AES128 key and encrypts a piece of data into a file. We use the EAX mode because it allows the receiver to detect any unauthorized modification (similarly, we could have used other authenticated encryption modes like GCM, CCM or SIV).

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

file_out = open("encrypted.bin", "wb")
key = get_random_bytes(16)
nonce = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX, nonce)
ciphertext, tag = cipher.encrypt_and_digest(data)
[ file_out.write(x) for x in (nonce, tag, ciphertext) ]
```

At the other end, the receiver can securely load the piece of data back (if they know the key!). Note that the code generates a `ValueError` exception when tampering is detected.

```
from Crypto.Cipher import AES

file_in = open("encrypted.bin", "rb")
nonce, tag, ciphertext = [ file_in.read(x) for x in (16, 16, -1) ]
# let's assume that the key is somehow available again
cipher = AES.new(key, AES.MODE_EAX, nonce)
data = cipher.decrypt_and_verify(ciphertext, tag)
```

5.2 Generate an RSA key

The following code generates a new RSA key pair (secret) and saves it into a file, protected by a password. We use the scrypt key derivation function to thwart dictionary attacks. At the end, the code prints our the RSA public key in ASCII/PEM format:

```
from Crypto.PublicKey import RSA

secret_code = "Unguessable"
key = RSA.generate(2048)
file_out = open("rsa_key.bin", "wb")
encrypted_key = key.exportKey(passphrase=secret_code, pkcs=8,
```

```
                                protection="scryptAndAES128-CBC")
file_out.write(encrypted_key)

print key.publickey().exportKey()
```

The following code reads the private RSA key back in, and then prints again the public key:

```
from Crypto.PublicKey import RSA

secret_code = "Unguessable"
file_in = open("rsa_key.bin", "rb")
key = RSA.importKey(file_in.read(), passphrase=secret_code)

print key.publickey().exportKey()
```

5.3 Encrypt data with RSA

The following code encrypts a piece of data for a receiver we have the RSA public key of. The RSA public key is stored in a file called `receiver.pem`.

Since we want to be able to encrypt an arbitrary amount of data, we use a hybrid encryption scheme. We use RSA with PKCS#1 OAEP for asymmetric encryption of an AES session key. The session key can then be used to encrypt all the actual data.

As in the first example, we use the EAX mode to allow detection of unauthorized modifications.

```
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP

file_out = open("encrypted_data.bin", "wb")

recipient_key = RSA.importKey(open("receiver.pem").read())
session_key = get_random_bytes(16)
nonce = get_random_bytes(16)

# Encrypt the session key with the public RSA key
cipher_rsa = PKCS1_OAEP.new(recipient_key)
file_out.write(cipher_rsa.encrypt(session_key))

# Encrypt the data with the AES session key
cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
ciphertext, tag = cipher_aes.encrypt_and_digest(data)
[ file_out.write(x) for x in (nonce, tag, ciphertext) ]
```

The receiver has the private RSA key. They will use it to decrypt the session key first, and with that the rest of the file:

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_OAEP
import math

file_in = open("encrypted_data.bin", "rb")

private_key = RSA.importKey(open("private.pem").read())
rsa_size = ceil(private_key.size()/8.0)

enc_session_key, nonce, tag, ciphertext = \
```

```
   [ file_in.read(x) for x in (rsa_size, 16, 16, -1) ]

# Decrypt the session key with the public RSA key
cipher_rsa = PKCS1_OAEP.new(private_key)
session_key = cipher_rsa.decrypt(enc_session_key)

# Decrypt the data with the AES session key
cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
data = cipher.decrypt_and_verify(ciphertext, tag)
```

CHAPTER 6

Contribute and support

- Do not be afraid to contribute with small and apparently insignificant improvements like correction to typos. Every change counts.
- Read carefully the License of PyCryptodome. By submitting your code, you acknowledge that you accept to release it according to the BSD 2-clause license.
- You must disclaim which parts of your code in your contribution were partially copied or derived from an existing source. Ensure that the original is licensed in a way compatible to the *BSD 2-clause license*.
- You can propose changes in any way you find most convenient. However, the preferred approach is to:
  - Clone the main repository on GitHub.
  - Create a branch and modify the code.
  - Send a pull request upstream with a meaningful description.
- Provide tests (in `Crypto.SelfTest`) along with code. If you fix a bug add a test that fails in the current version and passes with your change.
- If your change breaks backward compatibility, hightlight it and include a justification.
- Ensure that your code complies to PEP8 and PEP257.
- Ensure that your code does not use constructs or includes modules not present in Python 2.4.
- Add a short summary of the change to the file `Changelog.rst`.
- Add your name to the list of contributors in the file `AUTHORS.rst`.

The PyCryptodome mailing list is hosted on Google Groups. You can mail any comment or question to *pycryptodome@googlegroups.com*.

Bug reports can be filed on the GitHub tracker.

CHAPTER 7

Future plans

Future releases will include:

- Break-up test cases of ciphers and make them mode-specific
- Make all hash objects non-copiable and immutable after the first digest
- Automatic IV/nonce generation for cipher modes
- Move API documentation from epydoc to sphinx
- Implement AES with bitslicing
- Move old ciphers into a Museum submodule
- **Add algorithms:**
  - Poly1305
  - Elliptic Curves (ECDSA, ECIES, ECDH)
  - Camellia, GOST
  - Diffie-Hellman
  - bcrypt
  - SRP
- **Add more key management:**
  - Export/import of EC keys
  - Export/import of DSA domain parameters
  - JWK
- Add support for CMS/PKCS#7
- Add support for RNG backed by PKCS#11 and/or KMIP
- Add support for Format-Preserving Encryption
- Add the complete set of NIST test vectors for the various algorithms
- Remove dependency on libtomcrypto headers
- Speed up (T)DES with a bitsliced implementation
- Add support for PCLMULQDQ in AES-GCM
- Coverage testing

- Run lint on the C code
- Add (minimal) support for PGP
- Add (minimal) support for PKIX / X.509

CHAPTER 8

Changelog

8.1 3.3 (29 October 2015)

8.1.1 New features

- Windows wheels bundle the MPIR library
- Detection of faults occuring during secret RSA operations
- Detection of non-prime (weak) q value in DSA domain parameters
- Added original Keccak hash family (b=1600 only). In the process, simplified the C code base for SHA-3.
- Added SHAKE128 and SHAKE256 (of SHA-3 family)

8.1.2 Resolved issues

- GH#3: gcc 4.4.7 unhappy about double typedef

8.1.3 Breaks in compatibility

- Removed method *copy* from all SHA-3 hashes
- Removed ability to *update* a SHA-3 hash after the first call to *(hex)digest*

8.2 3.2.1 (9 September 2015)

8.2.1 New features

- Windows wheels are automatically built on Appveyor

8.3 3.2 (6 September 2015)

8.3.1 New features

- Added hash functions BLAKE2b and BLAKE2s.

- Added stream cipher ChaCha20.
- Added OCB cipher mode.
- CMAC raises an exception whenever the message length is found to be too large and the chance of collisions not negligeable.
- New attribute `oid` for Hash objects with ASN.1 Object ID
- Added `Crypto.Signature.pss` and `Crypto.Signature.pkcs1_15`
- Added NIST test vectors (roughly 1200) for PKCS#1 v1.5 and PSS signatures.

8.3.2 Resolved issues

- tomcrypt_macros.h asm error #1

8.3.3 Breaks in compatibility

- Removed keyword `verify_x509_cert` from module method `importKey` (RSA and DSA).
- Reverted to original PyCrypto behavior of method `verify` in `PKCS1_v1_5` and `PKCS1_PSS`.

8.4 3.1 (15 March 2015)

8.4.1 New features

- Speed up execution of Public Key algorithms on PyPy, when backed by the Gnu Multiprecision (GMP) library.
- GMP headers and static libraries are not required anymore at the time PyCryptodome is built. Instead, the code will automatically use the GMP dynamic library (.so/.DLL) if found in the system at runtime.
- Reduced the amount of C code by almost 40% (4700 lines). Modularized and simplified all code (C and Python) related to block ciphers. Pycryptodome is now free of CPython extensions.
- Add support for CI in Windows via Appveyor.
- RSA and DSA key generation more closely follows FIPS 186-4 (though it is not 100% compliant).

8.4.2 Resolved issues

- None

8.4.3 Breaks in compatibility

- New dependency on ctypes with Python 2.4.
- The `counter` parameter of a CTR mode cipher must be generated via `Crypto.Util.Counter`. It cannot be a generic callable anymore.
- Removed the `Crypto.Random.Fortuna` package (due to lack of test vectors).
- Removed the `Crypto.Hash.new` function.
- The `allow_wraparound` parameter of `Crypto.Util.Counter` is ignored. An exception is always generated if the counter is reused.

- `DSA.generate`, `RSA.generate` and `ElGamal.generate` do not accept the `progress_func` parameter anymore.
- Removed `Crypto.PublicKey.RSA.RSAImplementation`.
- Removed `Crypto.PublicKey.DSA.DSAImplementation`.
- Removed ambiguous method `size()` from RSA, DSA and ElGamal keys.

8.5 3.0 (24 June 2014)

8.5.1 New features

- Initial support for PyPy.
- SHA-3 hash family based on the April 2014 draft of FIPS 202. See modules `Crypto.Hash.SHA3_224/256/384/512`. Initial Keccak patch by Fabrizio Tarizzo.
- Salsa20 stream cipher. See module `Crypto.Cipher.Salsa20`. Patch by Fabrizio Tarizzo.
- Colin Percival's `scrypt` key derivation function (`Crypto.Protocol.KDF.scrypt`).
- Proper interface to FIPS 186-3 DSA. See module `Crypto.Signature.DSS`.
- Deterministic DSA (RFC6979). Again, see `Crypto.Signature.DSS`.
- HMAC-based Extract-and-Expand key derivation function (`Crypto.Protocol.KDF.HKDF`, RFC5869).
- Shamir's Secret Sharing protocol, compatible with *ssss* (128 bits only). See module `Crypto.Protocol.SecretSharing`.
- Ability to generate a DSA key given the domain parameters.
- Ability to test installation with a simple `python -m Crypto.SelfTest`.

8.5.2 Resolved issues

- LP#1193521: `mpz_powm_sec()` (and Python) crashed when modulus was odd.
- Benchmarks work again (they broke when ECB stopped working if an IV was passed. Patch by Richard Mitchell.
- LP#1178485: removed some catch-all exception handlers. Patch by Richard Mitchell.
- LP#1209399: Removal of Python wrappers caused HMAC to silently produce the wrong data with SHA-2 algorithms.
- LP#1279231: remove dead code that does nothing in SHA-2 hashes. Patch by Richard Mitchell.
- LP#1327081: AESNI code accesses memory beyond buffer end.
- Stricter checks on ciphertext and plaintext size for textbook RSA (kudos to sharego).

8.5.3 Breaks in compatibility

- Removed support for Python < 2.4.
- Removed the following methods from all 3 public key object types (RSA, DSA, ElGamal):
  - `sign`
  - `verify`

  - encrypt
  - decrypt
  - blind
  - unblind
  - can_encrypt
  - can_sign

  Code that uses such methods is doomed anyway. It should be fixed ASAP to use the algorithms available in Crypto.Signature and Crypto.Cipher.

- The 3 public key object types (RSA, DSA, ElGamal) are now unpickable.
- Symmetric ciphers do not have a default mode anymore (used to be ECB). An expression like AES.new(key) will now fail. If ECB is the desired mode, one has to explicitly use AES.new(key, AES.MODE_ECB).
- Unsuccessful verification of a signature will now raise an exception [reverted in 3.2].
- Removed the Crypto.Random.OSRNG package.
- Removed the Crypto.Util.winrandom module.
- Removed the Crypto.Random.randpool module.
- Removed the Crypto.Cipher.XOR module.
- Removed the Crypto.Protocol.AllOrNothing module.
- Removed the Crypto.Protocol.Chaffing module.
- Removed the parameters disabled_shortcut and overflow from Crypto.Util.Counter.new.

8.5.4 Other changes

- Crypto.Random stops being a userspace CSPRNG. It is now a pure wrapper over os.urandom.
- Added certain resistance against side-channel attacks for GHASH (GCM) and DSA.
- More test vectors for HMAC-RIPEMD-160.
- Update libtomcrypt headers and code to v1.17 (kudos to Richard Mitchell).
- RSA and DSA keys are checked for consistency as they are imported.
- Simplified build process by removing autoconf.
- Speed optimization to PBKDF2.
- Add support for MSVC.
- Replaced HMAC code with a BSD implementation. Clarified that starting from the fork, all contributions are released under the BSD license.

CHAPTER 9

License

The source code in PyCryptodome is partially in the public domain and partially released under the BSD 2-Clause license.

In either case, there are minimal if no restrictions on the redistribution, modification and usage of the software.

9.1 Public domain

All code originating from PyCrypto is free and unencumbered software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

In jurisdictions that recognize copyright laws, the author or authors of this software dedicate any and all copyright interest in the software to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

For more information, please refer to <http://unlicense.org>

9.2 BSD license

All direct contributions to PyCryptodome are released under the following license. The copyright of each piece belongs to the respective author.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9.3 OCB license

The OCB cipher mode is patended in US. The directory Doc/ocb contains three free licenses for implementors and users. As a general statement, OCB can be freely used for software not meant for military purposes. Contact your attorney for further information.

9.4 MPIR license

When distributed as a Windows wheel, Pycryptodome bundles an unmodified, binary version of the MPIR library (https:\www.mpir.org) which is licensed under the LGPLv3, a copy of which is available under Doc/mpir.