
PyCryptodome Documentation

Release 3.4.6

Legrandin

Jul 11, 2017

Contents

1	PyCryptodome	1
1.1	News	2
2	Features	3
3	Installation	7
3.1	Linux Ubuntu	8
3.2	Linux Fedora	8
3.3	Windows (pre-compiled)	8
3.4	Windows (from sources, Python 2.x, Python <=3.2)	9
3.5	Windows (from sources, Python 3.3 and 3.4)	10
3.6	Windows (from sources, Python 3.5 and newer)	11
3.7	PGP verification	11
4	API documentation	13
5	Examples	15
5.1	Encrypt data with AES	15
5.2	Generate an RSA key	16
5.3	Encrypt data with RSA	16
6	Contribute and support	19
7	Future plans	21
8	Changelog	23
8.1	3.4.6 (18 May 2017)	23
8.2	3.4.5 (6 February 2017)	23
8.3	3.4.4 (1 February 2017)	23
8.4	3.4.3 (17 October 2016)	24
8.5	3.4.2 (8 March 2016)	24
8.6	3.4.1 (21 February 2016)	24
8.7	3.4 (7 February 2016)	24
8.8	3.3.1 (1 November 2015)	25
8.9	3.3 (29 October 2015)	26
8.10	3.2.1 (9 September 2015)	26
8.11	3.2 (6 September 2015)	26

8.12	3.1 (15 March 2015)	27
8.13	3.0 (24 June 2014)	27
9	License	31
9.1	Public domain	31
9.2	BSD license	31
9.3	OCB license	32
9.4	MPIR license	32

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

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

The installation procedure depends on the package you want the library in. PyCryptodome can be used as:

1. **a drop-in replacement for the old PyCrypto library.** You install it with:

```
pip install pycryptodome
```

In this case, all modules are installed under the `Crypto` package.

One must avoid having both PyCrypto and PyCryptodome installed at the same time, as they will interfere with each other.

This option is therefore recommended only when you are sure that the whole application is deployed in a `virtualenv`.

2. **a library independent of the old PyCrypto.** You install it with:

```
pip install pycryptodomex
```

In this case, all modules are installed under the `Cryptodome` package. PyCrypto and PyCryptodome can coexist.

For faster public key operations, you should have [GMP](#) installed in your system (except on Windows, as the wheel on PyPi already comes bundled with the equivalent [MPIR](#) library).

PyCryptodome is a fork of PyCrypto. 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
- Elliptic curves cryptography (NIST P-256 curve only)

- Better and more compact API (*nonce* and *iv* attributes for ciphers, automatic generation of random nonces and IVs, simplified CTR cipher mode, and more)
- SHA-3 (including SHAKE XOFs) and BLAKE2 hash algorithms
- Salsa20 and ChaCha20 stream ciphers
- scrypt and HKDF
- Deterministic (EC)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
- Cleaner RSA and DSA key generation (largely based on FIPS 186-4)
- Major clean ups and simplification of the code base

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.

For more information, see the [homepage](#).

All the code can be downloaded from [GitHub](#).

News

- **17 May 2017 (NEW)**. Bugfix release 3.4.6.
- 6 Feb 2017. Bugfix release 3.4.5.
- 1 Feb 2017. Bugfix release 3.4.4.
- 17 Oct 2016. Bugfix release 3.4.3.
- 8 Mar 2016. Bugfix release 3.4.2.
- 21 Feb 2016. Release 3.4.1.
- 7 Feb 2016. Release 3.4.
- Nov 2015. Release 3.3.1.
- 29 Oct 2015. 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.

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 hashes (224, 256, 384, 512)
 - SHA-3 hashes (224, 256, 384, 512) and XOFs (SHAKE128, SHAKE256)
 - Keccak (original submission to SHA-3)
 - BLAKE2b and BLAKE2s
 - RIPE-MD160
 - MD5
- Message Authentication Codes (MAC):
 - HMAC
 - CMAC
- Asymmetric key generation:
 - RSA
 - DSA
 - ECC (NIST P-256 curve only)
 - 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
- Asymmetric digital signatures:
 - PKCS#1 (RSA)
 - * RSASSA-PKCS1-v1_5
 - * RSASSA-PSS
 - (EC)DSA
 - * Nonce-based (FIPS 186-3)
 - * Deterministic (RFC6979)

- Key derivation:
 - PBKDF1
 - PBKDF2
 - scrypt
 - HKDF
- Other cryptographic protocols:
 - Shamir Secret Sharing
 - Padding
 - * PKCS#7
 - * ISO-7816
 - * X.923

The installation procedure depends on the package you want the library in. PyCryptodome can be used as:

1. **a drop-in replacement for the old PyCrypto library.** You install it with:

```
pip install pycryptodome
```

In this case, all modules are installed under the `Crypto` package. You can test everything is right with:

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

One must avoid having both PyCrypto and PyCryptodome installed at the same time, as they will interfere with each other.

This option is therefore recommended only when you are sure that the whole application is deployed in a `virtualenv`.

2. **a library independent of the old PyCrypto.** You install it with:

```
pip install pycryptodomex
```

You can test everything is right with:

```
python -m Cryptodome.SelfTest
```

In this case, all modules are installed under the `Cryptodome` package. PyCrypto and PyCryptodome can coexist.

The procedures below go a bit more in detail, by explaining how to setup the environment for compiling the C extensions for each OS, and how to install the GMP library.

All instructions to follow install PyCryptodome as the `Cryptodome` package (option #2). Change `pycryptodomex` to `pycryptodome` if you prefer option #1 (`Crypto` package).

Linux Ubuntu

For Python 2.x:

```
$ sudo apt-get install build-essential libgmp3-dev python-dev
$ pip install pycryptodomex
$ python -m Cryptodome.SelfTest
```

For Python 3.x:

```
$ sudo apt-get install build-essential libgmp3-dev python3-dev
$ pip install pycryptodomex
$ python3 -m Cryptodome.SelfTest
```

For PyPy:

```
$ sudo apt-get install build-essential libgmp3-dev pypy-dev
$ pip install pycryptodomex
$ pypy -m Cryptodome.SelfTest
```

Linux Fedora

For Python 2.x:

```
$ sudo yum install gcc gmp python-devel
$ pip install pycryptodomex
$ python -m Cryptodome.SelfTest
```

For Python 3.x:

```
$ sudo yum install gcc gmp python3-devel
$ pip install pycryptodomex
$ python3 -m Cryptodome.SelfTest
```

For PyPy:

```
$ sudo yum install gcc gmp pypy-devel
$ pip install pycryptodomex
$ pypy -m Cryptodome.SelfTest
```

Windows (pre-compiled)

1. Install PyCryptodome as a [wheel](#):

```
> pip install pycryptodomex
```

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

```
> python -m Cryptodome.SelfTest
```

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. 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.

2. **[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.

3. **[Only once]** Install [Virtual Clone Drive](#).

4. **[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.

5. 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.

6. Compile and install PyCryptodome:

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

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

```
> python -m Cryptodome.SelfTest
```

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. 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.

2. **[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.

3. **[Only once]** Install [Virtual Clone Drive](#).

4. **[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.

5. 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.

6. Compile and install PyCryptodome:

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

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

```
> python -m Cryptodome.SelfTest
```

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 pycryptodomex --no-use-wheel
```

PGP verification

All source packages and wheels on PyPI are cryptographically signed. They can be verified with the following PGP key:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFTXjPgBEADc3j7vnma9MXRshBPPXXenVpthQD6lrF/3XaBT2RptSf/viOD+
tz85du5XVp+r0SYYGeMnJcQ9NsztXblN/lnKgkfwRmSrB+V6QGS+e3bR5d9OIxzN
7haPxBnyRj//hCT/kKis6fa7N9wtwKBBjbaSX+9vpt7Rrt203sKfcChA4iR3EG89
TNQoc/kGGmwk/gyjfU38726v0NOhMKJp2154iQQVZ76hTDk6GkOYHTcPxdkAj4jS
Dd74M9sOtO0lyDLHOLcWNnlWGgZjtZ0z0qSyFXRSuOfggTxrepWQgKWXZgVB4Jo
0bhmXPAV8vkX5BoG6zGkYb47NGGvknax6jCvFYTCp1sOmVt f5UTVKPplFm077tQg
0KZNAvEQrdWRIiQ1cCGCoF2ALex3VmVdefHOHnmyY7xAlzP0c8z1DsgZgMnytNn
GPusWeqQVi jRxenl+lyhbkb9ZLDq7mOkCRXsZe9J2+5aLTJbJu3+Wx6BEyNIHP/f
K3E77nXvC0oKaYtbTweQSBAggAXP+7oQaA0ea2SLO176xJdNfC51kQEtMMSZI4gN
iSjqUxXW2N5qEHHexlatmTtk4W9tQEw030a0UCxzDJMhD0aWFKq7wOxoCQ1q821R
vxBH4cfGwDL/1FUcuCMSulc6fhTM9pvMXgjdEXcoiLSTdaHuVLUqmF/E0wARAQAB
tB9MZwdyYW5kaW4gPghlBGRlcm1qc0BnbWFrpbC5jb20+iQI4BBMBAgAIBQJU14z4
AhsDBgsJCAcDAgYVCAIJCgsEFgIDAQIeAQIXgAAKCRDabO+N4RaZen7IEACpApha
vRwPB+Dv87aEyVmJfZ96Nb3mxHdeP2uSmUxAODzoB5oJJ1QL6HRxEV1U8idjdf73H
DX39ZC7izD+oYIve9sNwTbKqJCZaTx1TDdgSF1N57eJ0LELAy+SqpHtaMJPk7SfJ
l/iYoUYxByPLZU1wDwZEDNzt9RCGy3bd/vF/AxWjdUJJP3E4j5hswvIGSf8/Tp3
MDROU1BaNB0d0CLvBHok8/xavw06Dk/fe4hJhd5uZcEptd1GJcPq51z2yr7PGUcb
oERsKZyG8c9fd7j8qoTd6jMIW6fBVHdxIMxW6/Z45X/vVciQGSzEl/yjPUW42kyr
Ib6M16YmnDzp8b14NNFvvr9uWvOdUkep2Bi8s8kBMJ7G9rHHJcdVY/tP1ECS9Bse
hN4v5oJJ4v5mM/MiWRGKyKzULWklonpiq6CewYkmXQDMRnjGXhjCwRb6LuSIkIXd
gKvDNpJ8yEhAfmPvA4I3laMoof/tSZ7ZuyLSZGLKl6hoNIB13HCn4dnjnBeaXCWX
pThgeOWxV6u1fhz4CeC1Hc8WOYr8S7G8P10Ji6owOcj/a1QuCW8XDB2omCTX1hfj
zpc9dX8HgmUVnbPNiMjphihbKXoOcunRx4ZvqIa8mnTbI4tHtR0K0tI4MmbpcVOZ
8IFJOnzJXuZiL57ijLREisPYmHfBHAgmh1j/W7kCDQRU14z4ARAA3QATrgvOSYFh
nJOnIz6P03G9kXWjJ8wvp3yE1/PwwTc3NbVUSNCW14xgM2Ryhn9NVh8iEGtPGmUP
4vu7rvuLC2rBs1joBTYqf0mDghlZrb5ZjXv5LcG9SA6FdAXRU6T+b1G2ychKkhEh
```

```
d/ulLw/TKLds9zHhE+hkAagLQ5jqjcQN0iX5EYaOukiPUGmnd9foEGi9YMYtRdrH
+3bZxUpsRStLBWJ6auY7Bla8NJOhaWpr5p/ls+mnDwoqf+tXCCps1Da/pfHKYDFc
2VVdyM/VfNny9eaczYpnj5hvIAACWChgDBwxPh2DGdUfiQi/QqrK96+F7ulqz6V
2exX4CL0cPv5fUpQqSU/0R5WApM9b12+w1jFhoCXlydU9Hnn+0GatGzEoo3yrV/m
PXv7d6NdZxyOqgxu/ai/z++F2pWUXSBxZN3Gv28boFKQhmttTcFudNUTQOchhn8
Pf/ipVISqrsZorTx9Qx4fPScEWjwbh84Uz20bx0sQs1oYcek2YG5RhEdzqJ6W78R
S/dbz1NYMXGdkxB6C63m8oiGvw0hdN/iGVqpNAoldFmJnFqSgKpyPwfLmmdstJ6f
xFZdGPNkexCpHbKr9fg50jZRenIGai79qPiEtCZHIdpeemSrc7TKRPV3H2aMNFg
L5HTqcyam2+QrMtHPMoOFzckigLimMAEQEAAYkCHwQYAIACQUCVNeM+AIbDAAK
CRDabO+N4RaZEo71D/45J6z2wbL8aIudGEL0aY3hfmW3qrUyoHgaw35KsOY9vZwb
cZuJe0RlYptOreH/NrbR5SXODfhd2sxYyyvXB0uZh9i700BsrAd5UE01GCvToPwh
7IpMV3GSSAB4P8XyJh20tZqiZOYKhmbf29gUDzqAI6GzUa0U8xidUKpW2zqYGZjp
wk3RI1fs7tyi/0N8B9tIZF48kbvpFDAjF8w7NSCrgRquAL7zJZIG5o5zXJM/ffF3
67Dnz278MbifdM/HJ+Tj0R0UvVki9Z61nT653SoUgviLQyC72XI+x0+3GQwsE38a
5aJNZ1NBD3/v+gERQxRfhM5iLFLXK0Xe4K2XFM1g0yN4L4bQPbhSCq88g9Dhmygk
XPbBsrK0NKPvnyGyUXM0VpgRbot11hxx02jC3HxS1n1LF+oQdkKFzJAMOU7UbpX/
oO+286J1FmpG+fiHibvp1Quq48immntzTeLZbYCsG4mrM+ySYd0Er0G8TBdAOTiN
3zMbGX0QOO2fOsJ1d980cVjHn5CbAo8C0A/4/R2cXAfpacbvTinQ5BVk9Nka2dNb
kmnTStP2qILWmm5ASXlWhOjWNmptvsUcK+8T+uQboLioEv190b4j5Irs/OpOuP0K
v4woCi9+03HMS42qGSe/igClFO3+gUMZg9PJnTJhuaTbytXhUBgBRUPsS+lQAQ==
=DpoI
-----END PGP PUBLIC KEY BLOCK-----
```


CHAPTER 4

API documentation

The API can be found [here](#). Soon it will be moved on these pages.

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

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(data)

file_out = open("encrypted.bin", "wb")
[ file_out.write(x) for x in (cipher.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)
```

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 out the RSA public key in ASCII/PEM format:

```
from Crypto.PublicKey import RSA

secret_code = "Unguessable"
key = RSA.generate(2048)
encrypted_key = key.exportKey(passphrase=secret_code, pkcs=8,
                              protection="scryptAndAES128-CBC")

file_out = open("rsa_key.bin", "wb")
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"
encoded_key = open("rsa_key.bin", "rb").read()
key = RSA.import_key(encoded_key, passphrase=secret_code)

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

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.import_key(open("receiver.pem").read())
session_key = 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)
ciphertext, tag = cipher_aes.encrypt_and_digest(data)
[ file_out.write(x) for x in (cipher.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

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

private_key = RSA.import_key(open("private.pem").read())

enc_session_key, nonce, tag, ciphertext = \
    [ file_in.read(x) for x in (private_key.size_in_bytes(), 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)
```

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, highlight 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 `py-cryptodome@googlegroups.com`.

Bug reports can be filed on the [GitHub tracker](#).

Future releases will include:

- Make all hash objects non-copiable and immutable after the first digest
- Add alias 'segment_bits' to parameter 'segment_size' for CFB
- Coverage testing
- Add support for memoryview/buffer interface
- Move API documentation from epydoc to sphinx
- Implement AES with bitslicing
- Add unit tests for PEM I/O
- Move old ciphers into a Museum submodule
- Add more ECC curves
- Import/export of ECC keys with compressed points
- **Add algorithms:**
 - Poly1305
 - Elliptic Curves (ECIES, ECDH)
 - Camellia, GOST
 - Diffie-Hellman
 - bcrypt
 - argon2
 - SRP
- **Add more key management:**
 - 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
- Remove dependency on libtomcrypto headers
- Speed up (T)DES with a bitsliced implementation
- Add support for PCLMULQDQ in AES-GCM
- Run lint on the C code
- Add (minimal) support for PGP
- Add (minimal) support for PKIX / X.509

3.4.6 (18 May 2017)

Resolved issues

- GH#65. Keccak, SHA3, SHAKE and the seek functionality for ChaCha20 were not working on big endian machines. Fixed. Thanks to Mike Gilbert.
- A few fixes in the documentation.

3.4.5 (6 February 2017)

Resolved issues

- The library can also be compiled using MinGW.

3.4.4 (1 February 2017)

Resolved issues

- Removed use of `alloca()`.
- [Security] Removed implementation of deprecated “quick check” feature of PGP block cipher mode.
- Improved the performance of `script` by converting some Python to C.

3.4.3 (17 October 2016)

Resolved issues

- Undefined warning was raised with libgmp version < 5
- Forgot inclusion of `alloca.h`
- Fixed a warning about type mismatch raised by recent versions of `ffi`

3.4.2 (8 March 2016)

Resolved issues

- Fix renaming of package for `install` command.

3.4.1 (21 February 2016)

New features

- Added option to install the library under the `Cryptodome` package (instead of `Crypto`).

3.4 (7 February 2016)

New features

- Added `Crypto.PublicKey.ECC` module (NIST P-256 curve only), including export/import of ECC keys.
- Added support for ECDSA (FIPS 186-3 and RFC6979).
- For CBC/CFB/OFB/CTR cipher objects, `encrypt()` and `decrypt()` cannot be intermixed.
- CBC/CFB/OFB, the cipher objects have both `IV` and `iv` attributes. `new()` accepts `IV` as well as `iv` as parameter.
- For CFB/OPENPGP cipher object, `encrypt()` and `decrypt()` do not require the plaintext or ciphertext pieces to have length multiple of the CFB segment size.
- Added dedicated tests for all cipher modes, including NIST test vectors
- CTR/CCM/EAX/GCM/SIV/Salsa20/ChaCha20 objects expose the `nonce` attribute.
- For performance reasons, CCM cipher optionally accepted a pre-declaration of the length of the associated data, but never checked if the actual data passed to the cipher really matched that length. Such check is now enforced.
- CTR cipher objects accept parameter `nonce` and possibly `initial_value` in alternative to `counter` (which is deprecated).
- All `iv/IV` and `nonce` parameters are optional. If not provided, they will be randomly generated (exception: `nonce` for CTR mode in case of block sizes smaller than 16 bytes).
- Refactored ARC2 cipher.
- Added `Crypto.Cipher.DES3.adjust_key_parity()` function.

- Added `RSA.import_key` as an alias to the deprecated `RSA.importKey` (same for the DSA module).
- Added `size_in_bits()` and `size_in_bytes()` methods to `RsaKey`.

Resolved issues

- RSA key size is now returned correctly in `RsaKey.__repr__()` method (kudos to *hannesv*).
- CTR mode does not modify anymore `counter` parameter passed to `new()` method.
- CTR raises `OverflowError` instead of `ValueError` when the counter wraps around.
- PEM files with Windows newlines could not be imported.
- `Crypto.IO.PEM` and `Crypto.IO.PKCS8` used to accept empty passphrases.
- GH#6: `NotImplementedError` now raised for unsupported methods `sign`, `verify`, `encrypt`, `decrypt`, `blind`, `unblind` and `size` in objects `RsaKey`, `DsaKey`, `ElGamalKey`.

Breaks in compatibility

- Parameter `segment_size` cannot be 0 for the CFB mode.
- For OCB ciphers, a final call without parameters to `encrypt` must end a sequence of calls to `encrypt` with data (similarly for `decrypt`).
- Key size for ARC2, ARC4 and Blowfish must be at least 40 bits long (still very weak).
- DES3 (Triple DES module) does not allow keys that degenerate to Single DES.
- Removed method `getRandomNumber` in `Crypto.Util.number`.
- Removed module `Crypto.pct_warnings`.
- Removed attribute `Crypto.PublicKey.RSA.algorithmIdentifier`.

3.3.1 (1 November 2015)

New features

- Opt-in for `update()` after `digest()` for SHA-3, keccak, BLAKE2 hashes

Resolved issues

- Removed unused SHA-3 and keccak test vectors, therefore significantly reducing the package from 13MB to 3MB.

Breaks in compatibility

- Removed method `copy()` from BLAKE2 hashes
- Removed ability to `update()` a BLAKE2 hash after the first call to `(hex)digest()`

3.3 (29 October 2015)

New features

- Windows wheels bundle the MPIR library
- Detection of faults occurring 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)

Resolved issues

- GH#3: gcc 4.4.7 unhappy about double typedef

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()`

3.2.1 (9 September 2015)

New features

- Windows wheels are automatically built on Appveyor

3.2 (6 September 2015)

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 negligible.
- 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.

Resolved issues

- `tomcrypt_macros.h` asm error #1

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`.

3.1 (15 March 2015)

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).

Resolved issues

- None

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.

3.0 (24 June 2014)

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`.

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).

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`

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`.

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.

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.

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>>

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.

OCB license

The OCB cipher mode is patented 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.

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.