

RSA

[RSA](#) is one of the most widespread and public key algorithms. Its security is based on the difficulty of factoring large integers. The algorithm has withstood attacks for more than 30 years, and it is therefore considered reasonably secure.

However, for new designs, it is recommended to use [ECC](#), because keys are smaller and private key operations are faster.

The RSA algorithm can be used for both confidentiality (encryption) and authentication (digital signature). Signing and decryption are significantly slower than verification and encryption.

The cryptographic strength is primarily linked to the length of the RSA modulus n . In 2023, a sufficient length is deemed to be 3072 bits. For more information, see the most recent [NIST](#) report. Both RSA ciphertexts and RSA signatures are as large as the RSA modulus n (384 bytes if n is 3072 bit long).

With this module you can generate new RSA keys:

```
>>> from Crypto.PublicKey import RSA
>>>
>>> mykey = RSA.generate(3072)
```

export an RSA private key and protect it with a password, so that it is resistant to brute force attacks:

```
>>> pwd = b'secret'
>>> with open("myprivatekey.pem", "wb") as f:
>>>     data = mykey.export_key(passphrase=pwd,
>>>                             pkcs=8,
>>>                             protection='PBKDF2WithHMAC-SHA512AndAES256-CBC',
>>>                             prot_params={'iteration_count':131072})
>>>     f.write(data)
```

and reimport it later:

```
>>> pwd = b'secret'
>>> with open("myprivatekey.pem", "rb") as f:
>>>     data = f.read()
>>>     mykey = RSA.import_key(data, pwd)
```

You can also export the public key, which is not sensitive:

```
>>> with open("mypublickey.pem", "wb") as f:
>>>     data = mykey.public_key().export_key()
```

For signing data with RSA, use a higher level module such as [PKCS#1 PSS \(RSA\)](#).

For encrypting data with RSA, use [PKCS#1 OAEP \(RSA\)](#).

class `Crypto.PublicKey.RSA.RsaKey`(**kwargs)

Class defining an RSA key, private or public. Do not instantiate directly. Use `generate()`, `construct()` or `import_key()` instead.

- Variables::**
- **n** (*integer*) – RSA modulus
 - **e** (*integer*) – RSA public exponent
 - **d** (*integer*) – RSA private exponent
 - **p** (*integer*) – First factor of the RSA modulus
 - **q** (*integer*) – Second factor of the RSA modulus
 - **invp** (*integer*) – Chinese remainder component ($p^{-1} \bmod q$)
 - **invq** (*integer*) – Chinese remainder component ($q^{-1} \bmod p$)
 - **u** (*integer*) – Same as `invp`


export_key(format='PEM', passphrase=None, pkcs=1, protection=None, randfunc=None, prot_params=None)

Export this RSA key.

Keyword Arguments::

- **format** (*string*) –

The desired output format:

- `'PEM'` (default) Text output, according to [RFC1421/RFC1423](#).
- `'DER'` Binary output.
- `'OpenSSH'` Text output, according to the OpenSSH specification. Only suitable for public keys ( [latest](#) keys).

Note that PEM contains a DER structure.

- **passphrase** (*bytes or string*) – (Private keys only) The passphrase to protect the private key.
- **pkcs** (*integer*) – (Private keys only) The standard to use for serializing the key: PKCS#1 or PKCS#8.
With `pkcs=1` (default), the private key is encoded with a simple **PKCS#1** structure (`RSAPrivateKey`). The key cannot be securely encrypted.
With `pkcs=8`, the private key is encoded with a **PKCS#8** structure (`PrivateKeyInfo`). PKCS#8 offers the best ways to securely encrypt the key.

! Note

This parameter is ignored for a public key. For DER and PEM, the output is always an ASN.1 DER


`SubjectPublicKeyInfo` structure.

- **protection** (*string*) – (For private keys only) The encryption scheme to use for protecting the private key using the passphrase. You can only specify a value if `pkcs=8`. For all possible protection schemes, refer to [the encryption parameters of PKCS#8](#). The recommended value is `'PBKDF2WithHMAC-SHA512AndAES256-CBC'`.

If `None` (default), the behavior depends on `format`:

- if `format='PEM'`, the obsolete PEM encryption scheme is used. It is based on MD5 for key derivation, and 3DES for encryption.
- if `format='DER'`, the `'PBKDF2WithHMAC-SHA1AndDES-EDE3-CBC'` scheme is used.

- **prot_params** (*dict*) – (For private keys only)
The parameters to use to derive the encryption key from the passphrase. `'protection'` must be also specified. For all possible values, refer to [the encryption parameters of PKCS#8](#). The recommendation is to use `{'iteration_count':21000}` for PBKDF2, and `{'iteration_count':131072}` for scrypt.

- **randfunc** (*callable*) – A function that provides random bytes. Only used for PEM encoding. The default is `Crypto.Random.get_random_bytes()`.  latest

Returns:: the encoded key
Return type:: bytes
Raises:: **ValueError** – when the format is unknown or when you try to encrypt a private key with *DER* format and PKCS#1.

⚠ Warning

If you don't provide a pass phrase, the private key will be exported in the clear!

has_private()

Whether this is an RSA private key

public_key()

A matching RSA public key.

Returns:: a new `RsaKey` object

size_in_bits()

Size of the RSA modulus in bits

size_in_bytes()

The minimal amount of bytes that can hold the RSA modulus

Crypto.PublicKey.RSA.construct(*rsa_components*, *consistency_check=True*)

Construct an RSA key from a tuple of valid RSA components.

The modulus *n* must be the product of two primes. The public exponent *e* must be odd and larger than 1.

In case of a private key, the following equations must apply:

$$\begin{aligned}p * q &= n \\e * d &\equiv 1 \pmod{\text{lcm}[(p - 1)(q - 1)]} \\p * u &\equiv 1 \pmod{q}\end{aligned}$$

Parameters:: **rsa_components** (*tuple*) –

A tuple of integers, with at least 2 and no more than 6 items. The items come in the following order:

1. RSA modulus *n*.
2. Public exponent *e*.

3. Private exponent d . Only required if the key is private.
4. First factor of n (p). Optional, but the other factor q must also be present.
5. Second factor of n (q). Optional.
6. CRT coefficient q , that is $p^{-1} \bmod q$. Optional.

Keyword Arguments:: **consistency_check** (*boolean*) – If `True`, the library will verify that the provided components fulfil the main RSA properties.

Raises:: **ValueError** – when the key being imported fails the most basic RSA validity checks.

Returns: An RSA key object (`RsaKey`).

`Crypto.PublicKey.RSA.generate(bits, randfunc=None, e=65537)`

Create a new RSA key pair.

The algorithm closely follows NIST [FIPS 186-4](#) in its sections B.3.1 and B.3.3. The modulus is the product of two non-strong probable primes. Each prime passes a suitable number of Miller-Rabin tests with random bases and a single Lucas test.

Parameters:: **bits** (*integer*) – Key length, or size (in bits) of the RSA modulus. It must be at least 1024, but **2048 is recommended**. The FIPS standard only defines 1024, 2048 and 3072.

Keyword Arguments::

- **randfunc** (*callable*) – Function that returns random bytes. The default is `Crypto.Random.get_random_bytes()`.
- **e** (*integer*) – Public RSA exponent. It must be an odd positive integer. It is typically a small number with very few ones in its binary representation. The FIPS standard requires the public exponent to be at least 65537 (the default).

Returns: an RSA key object (`RsaKey`), with private key).


`Crypto.PublicKey.RSA.import_key(extern_key, passphrase=None)`

Import an RSA key (public or private).

Parameters::

- **extern_key** (*string or byte string*) – The RSA key to import.

The following formats are supported for an RSA **public key**:

- X.509 certificate (binary or PEM format)
- X.509 `subjectPublicKeyInfo` DER SEQUENCE (binary or  [latest](#) encoding)
- **PKCS#1** `RSAPublicKey` DER SEQUENCE (binary or PEM encoding)

- An OpenSSH line (e.g. the content of `~/.ssh/id_ecdsa`, ASCII)

The following formats are supported for an RSA **private key**:

- PKCS#1 `RSAPrivateKey` DER SEQUENCE (binary or PEM encoding)
- PKCS#8 `PrivateKeyInfo` or `EncryptedPrivateKeyInfo` DER SEQUENCE (binary or PEM encoding)
- OpenSSH (text format, introduced in [OpenSSH 6.5](#))

For details about the PEM encoding, see [RFC1421/RFC1423](#).

- **passphrase** (*string or byte string*) – For private keys only, the pass phrase that encrypts the key.

Returns: An RSA key object (`RsaKey`).

Raises:: `ValueError/IndexError/TypeError` – When the given key cannot be parsed (possibly because the pass phrase is wrong).

Crypto.PublicKey.RSA.oid= `'1.2.840.113549.1.1.1'`

[Object ID](#) for the RSA encryption algorithm. This OID often indicates a generic RSA key, even when such key will be actually used for digital signatures.