pyage Documentation

Release 0.1.0

Kushal Das <mail@kushaldas.in>

Jul 13, 2020

Contents:

age is a simple, secure and modern encryption tool with small explicit keys, no config options, and UNIX-style composability. The format specification is at age-encryption.org/v1.

rage is a Rust implementation of the age tool. It is pronounced like the Japanese (with a hard g).

To discuss the spec or other age related topics, please email the mailing list at age-dev@googlegroups.com. age was designed by @Benjojo12 and @FiloSottile.

The reference interoperable Golang implementation is available at filippo.io/age.

This particular Python module is implemented on top of the Rust crate age.

CHAPTER 1

API Documentation

The name of the module is *pyage*. It provides the following functions for various operations.

1.1 create_newkey()

pyage.create_newkey() method returns a tuple of a new keypair as (publickey, secretkey).

```
>>> public, secret = pyage.create_newkey()
```

Note: Remember to save both the public key and the secret key in proper places.

1.2 encrypt_bytes()

pyage.encrypt_bytes() takes three arguments, first argument is the data to be encrypted in bytes, and a list of public keys, and an optional *armor*, and it returns encrypted text as bytes, or ascii-armored value if you pass *armor=True*.

```
>>> data = b"The secret text"
>>> key = "age1spp8yf63x4xu7l5esxlnzldwgmaaqmwrjw38vra9s7hw63pyzpqsq82gst"
>>> encrypted_btyes = pyage.encrypt_bytes(data, [key,]) # For raw bytes

>>> encrypted_btyes = pyage.encrypt_bytes(data, [key,], armor=True) # For ascii␣
↪armored output
>>> print(encrypted_btyes.decode("utf-8"))
-----BEGIN AGE ENCRYPTED FILE-----
YWdlLWVuY3J5cHRpb24ub3JnL3YxCi0+IFgyNTUxOSArZUlJS2NMbzJIK3NEeGlm
UXRaVkZCYzlGSytueitreDlJbWtwbWxQWmlFClBVT1h6djYvYjdnR3htZlpFbnNj
b09WMjFPbGFXUnF1OGJnYmRnN0hPU3cKLT4gam9pbnQtb2lsLWJmNiAyMCVPRgpQ
SVYrc1M0YkljVkVTYVhsNzJkM3owU2h5b01HY1VVMkxuOW5JTUpEWFlQdCt4UFdR
cmFqY2JacHBXdm96OXpyCi8wWm9URzRRWC9sak5XdFh4VDJiT0ZtYwotLS0gMUxB
```

(continues on next page)

(continued from previous page)

```
SFNUVHJCVHFaZGhNQVRuNW5WemRtc2JGNlpwR1dlWUlERW82SHdlZwpgo/1tH1V9
oM1Iw5goNrm9DBYb83lhxFDekLl7p/MppLnReibUfkmEqf12zO8BkQ==
-----END AGE ENCRYPTED FILE-----
```

1.3 encrypt_bytes_withpassword()

pyage.encrypt_bytes_withpassword() takes three arguments, first argument is the data to be encrypted in bytes, and the password, and an optional *armor*, and it returns encrypted text as bytes, or ascii-armored value if you pass *armor=True*.

```
>>> data = b"The secret text"
>>> password = "redhat"
>>> encrypted_btyes = pyage.encrypt_bytes_withpassword(data, password) # For raw bytes
```

1.4 encrypt_file()

pyage.encrypt_file() takes 4 arguments, first the input file path as str, the output file path (as str) is the second argument, and the 3rd argument is the list of public keys, the 4th and the last optional argument is for *armor* output. If you pass *True*, then it will created ascii armored output.

```
>>> key = "age1spp8yf63x4xu7l5esxlnzldwgmaaqmwrjw38vra9s7hw63pyzpqsq82gst"
>>> assert pyage.encrypt_file("/etc/hosts", "/tmp/hosts.age", [key,])
```

1.5 encrypt_file_withpassword()

pyage.encrypt_file() takes 4 arguments, first the input file path as str, the output file path (as str) is the second argument, and the 3rd argument is the password as string, the 4th and the last optional argument is for *armor* output. If you pass *True*, then it will created ascii armored output.

```
>>> password = "redhat"
>>> assert pyage.encrypt_file("/etc/hosts", "/tmp/hosts.age", password, armor=True)
```

1.6 decrypt_bytes()

pyage.decrypt_bytes() takes two arguments, first argument is the encrypted data as bytes, and then the secret key as string. It returns the decrypted bytes on success.

```
>>> secret = "AGE-SECRET-KEY-
↪19Z8Q85A9RTCLJ36EXCCCX0R6PTL0RPJ93JUZW48H7FLRJMSTV32S5XY2FA"
>>> decrypted_bytes = pyage.decrypt_bytes(encrypt_btyes, secret)
```

1.7 decrypt_bytes_withpassword()

pyage.decrypt_bytes() takes two arguments, first argument is the encrypted data as bytes, and then the password as string. It returns the decrypted bytes on success.

```
>>> password = "redhat"
>>> decrypted_bytes = pyage.decrypt_bytes_withpassword(encrypt_btyes, password)
```

1.8 decrypt_file()

pyage.decrypt_file() takes 3 arguments, first the encrypted file path, then new decrypted output filepath, and then secret key. Returns *True* in case of success.

```
>>> secret = "AGE-SECRET-KEY-
↪19Z8Q85A9RTCLJ36EXCCCX0R6PTL0RPJ93JUZW48H7FLRJMSTV32S5XY2FA"
>>> assert pyage.decrypt_file("/tmp/host.age", "/tmp/hosts", secret)
```

1.9 decrypt_file_withpassword()

pyage.decrypt_file() takes 3 arguments, first the encrypted file path, then new decrypted output filepath, and then password. Returns *True* in case of success.

```
>>> password = "redhat"
>>> assert pyage.decrypt_file("/tmp/host.age", "/tmp/hosts", password)
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search