drysponge Documentation

Sebastien Riou <matic@nimp.co.uk>

Mar 10, 2021

Contents

`drysponge` is the reference package for DryGASCON.

Other pages (online)

- project page on GitHub
- Download Page with releases
- This page, when viewed online is at https://drysponge.readthedocs.io/en/latest/

Installation

This installs a package that can be used from Python (`import drysponge`) or in command line (`python3 -m drysponge.drygascon128_aead`).

1.1 From PyPI

To install for the current user:

```
python3 -m pip install --user drysponge
```

To install for all users on the system, administrator rights (root) may be required.

```
python3 -m pip install drysponge
```

CLI examples

2.1 AEAD encryption

DryGASCON128k16:

```
$ python3 -m drysponge.drygascon128_aead e 000102030405060708090A0B0C0D0E0F␣
↪000102030405060708090A0B0C0D0E0F "" ""
BB857CC1CB30BD12F67FBBCC00206053
```

DryGASCON128k32:

```
$ python3 -m drysponge.drygascon128_aead e␣
↪000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F␣
↪000102030405060708090A0B0C0D0E0F "" ""
4F072157A92BF38F845B7C56DFC45042
```

DryGASCON128k56:

```
$ python3 -m drysponge.drygascon128_aead e 000102030405060708090A0B0C0D0E0F10111213141516
↪1718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031323334353637 0001020304050607080 9
↪0A0B0C0D0E0F "" ""
28830FE67DE9772201D254ABE4C9788D
```

2.2 AEAD decryption

Encryption of empty message returns a TAG:

```
$ python3 -m drysponge.drygascon128_aead e 000102030405060708090A0B0C0D0E0F␣
↪000102030405060708090A0B0C0D0E0F "" ""
BB857CC1CB30BD12F67FBBCC00206053
```

Decryption of that TAG returns empty message:

```
$ python3 -m drysponge.drygascon128_aead d 000102030405060708090A0B0C0D0E0F␣
↪000102030405060708090A0B0C0D0E0F "BB857CC1CB30BD12F67FBBCC00206053" ""

$
```

Decryption of that TAG with a single bit changed is detected:

```
$ python3 -m drysponge.drygascon128_aead d 000102030405060708090A0B0C0D0E0F␣
↪000102030405060708090A0B0C0D0E0F "BB857CC1CB30BD12F67FBBCC00206054" ""
Traceback (most recent call last):
  File "/usr/lib/python3.8/runpy.py", line 193, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "/usr/lib/python3.8/runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "/home/user/.local/lib/python3.8/site-packages/drysponge/drygascon128_aead.py",␣
↪line 6, in <module>
    aead(impl)
```

(continues on next page)

(continued from previous page)

```
  File "/home/user/.local/lib/python3.8/site-packages/drysponge/aead.py", line 45, in␣
↪aead
    out = impl.decrypt(key,nonce,message,associatedData,staticData)
  File "/home/user/.local/lib/python3.8/site-packages/drysponge/base/__init__.py", line␣
↪498, in decrypt
    raise ValueError('authentication tags different')
ValueError: authentication tags different
$
```

Decryption with display of some internal variable (for development purposes):

Last argument is the verbosity level: (0 to 5)

```
$ python3 -m drysponge.drygascon128_aead d 000102030405060708090A0B0C0D0E0F␣
↪000102030405060708090A0B0C0D0E0F "BB857CC1CB30BD12F67FBBCC00206053" "" 2
Decrypting 16 bytes ciphertext with 0 bytes of associated data and 0 bytes of static data
Key size: 16 bytes
   Key:             000102030405060708090A0B0C0D0E0F
   Nonce:           000102030405060708090A0B0C0D0E0F
   CipherText Tag:  BB857CC1CB30BD12F67FBBCC00206053
   F/G entry 0 (F with DS): padded=0, domain=1, finalize=1
   C[ 0] = 000102030405060708090A0B0C0D0E0F10FEFEFE8AF2F2F2FA171D872CB43FDF
   C[ 1] = 90C1F2A364656667
   X[ 0] = FA8EC1869CD9166FD9293729F9181919
       R = 00000000000000000000000000000000
       I = 000102030405060708090A0B0C0D0E0F
   Final state:
   C[ 0] = 4F14612C10DEAE707EE48FA416E3EC7299ABFEF803E8858AD0D3D2DB2ACB1CE0
   C[ 1] = CC9AB5DF6789B53B
   X[ 0] = FA8EC1869CD9166FD9293729F9181919
       R = BB857CC1CB30BD12F67FBBCC00206053
   Message:
```

2.3 Hash

Basic usage: it returns only the digest of input argument

```
$ python3 -m drysponge.drygascon128_hash ""
1EDC77386E20A37C721D6E77ADABB9C4830F199F5ED25284A13C1D84B9FC257A
$ python3 -m drysponge.drygascon128_hash 1234
3ABDC10FB9D6C5C82C87BFA0E356F0B01E68F31DF95CC5B7EADA142009FFF40C
$ python3 -m drysponge.drygascon128_hash 00
1BEC89506E75D725BF93BCCFDD6EC81DF05CA281CF5201E3EE0865A7063763EE
$ python3 -m drysponge.drygascon128_hash 01
2DF6DADE10483642F407ED281A3D703B431AEE11175ADDE2E33C67CC3174A176
$ python3 -m drysponge.drygascon128_hash 0102
3A2FC64FD2FE7F4057AC1BF13A7C5CE820447F123BFD286B7F5FEEF04CD7CABB
```

Developper usage: second input argument is verbosity level.

Repeating the last one with increased verbosity level: (0 to 5)

```
$ python3 -m drysponge.drygascon128_hash 0102 2
Hashing 2 bytes message: 0102
   Padded Message:  01020100000000000000000000000000
   F/G entry 0 (F with DS): padded=1, domain=2, finalize=1
   C[ 0] = 243F6A8885A308D313198A2E03707344243F6A8885A308D313198A2E03707344
   C[ 1] = 243F6A8885A308D3
   X[ 0] = A4093822299F31D0082EFA98EC4E6C89
```

(continues on next page)

(continued from previous page)

```
       R = 00000000000000000000000000000000
       I = 01020100000000000000000000000000
   F/G entry 1 (G):
   C[ 0] = 1F37D39F5B747A29297C046B2CDA8A87BB44A1D659D443C63FD459D7AAE7088B
   C[ 1] = 10653C489074148B
   X[ 0] = A4093822299F31D0082EFA98EC4E6C89
       R = 3A2FC64FD2FE7F4057AC1BF13A7C5CE8
   Final state:
   C[ 0] = 8C5E2A5F0D20BE0B52C044A4A439465CAD9E37560764D98A6D3E9E20AF357346
   C[ 1] = D9B474B1063DF323
   X[ 0] = A4093822299F31D0082EFA98EC4E6C89
       R = 20447F123BFD286B7F5FEEF04CD7CABB
   Digest: 3A2FC64FD2FE7F4057AC1BF13A7C5CE820447F123BFD286B7F5FEEF04CD7CABB
3A2FC64FD2FE7F4057AC1BF13A7C5CE820447F123BFD286B7F5FEEF04CD7CABB
```

Python3 examples

3.1 Creating an instance

First you need to create an instance of the cipher you want to use. If you target 256 bit level, use this:

```
from drysponge.drygascon import DryGascon
cipher128 = DryGascon.DryGascon128().instance()
```

If you target 256 bit level, use that:

```
from drysponge.drygascon import DryGascon
cipher256 = DryGascon.DryGascon256().instance()
```

3.2 Hash

```
import binascii
digest = cipher128.hash("abc".encode('utf-8'))
print(binascii.hexlify(digest))
```

```
b'8883ff9cf556714116390fda08768463cf82b12021d4697250d959d53aaa87cf'
```

3.3 Encrypt

```
ciphertext = cipher128.encrypt(
        key=binascii.unhexlify(b'000102030405060708090A0B0C0D0E0F'),
        nonce=binascii.unhexlify(b'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'),
        message="message".encode('utf-8'),
        associated_data="associated data".encode('utf-8'))
print(binascii.hexlify(ciphertext))
```

```
b'8c49773e01cf469eb668ae9908201262d8f14682800d52'
```

3.4 Decrypt

```
message = cipher128.decrypt(
        key=binascii.unhexlify(b'000102030405060708090A0B0C0D0E0F'),
        nonce=binascii.unhexlify(b'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'),
        ciphertext=ciphertext,
        associated_data="associated data".encode('utf-8'))
print(message)
```

```
bytearray(b'message')
```

3.5 Getting instance's parameter

```
print("'small' profile key size:\t", cipher128.key_size())
print("'fast' profile key size:\t", cipher128.fastkey_size())
print("'full' profile key size:\t", cipher128.fullkey_size())
print("nonce size:\t", cipher128.nonce_size())
print("block size:\t", cipher128.block_size())
print("tag size:\t", cipher128.tag_size())
```

```
'small' profile key size:    16
'fast' profile key size:     32
'full' profile key size:     56
nonce size:  16
block size:  16
tag size:    16
```

3.6 Using the SPY

Each instance as an independant verbosity level. By default operations are silent, that's usually what you want if you use the cipher in an application. If you wish to see the internal values, you can set the verbosity level to one of the following values:

- cipher128.SPY_ALG_IO: operation inputs/outputs (min verbosity bar none)
- cipher128.SPY_F_IO: F function's inputs/outputs level
- cipher128.SPY_ROUND_IO: GASCON function's inputs/outputs level
- cipher128.SPY_FULL: all intermediate values (max verbosity)

```
cipher128.Verbose(cipher128.SPY_ALG_IO)
cipher128.encrypt(
        key=binascii.unhexlify(b'000102030405060708090A0B0C0D0E0F'),
        nonce=binascii.unhexlify(b'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'),
        message="message".encode('utf-8'),
        associated_data="associated data".encode('utf-8'))
```

```
Encrypting 7 bytes message with 15 bytes of associated data and 0 bytes of static data
Key size: 16 bytes
   Key:             000102030405060708090A0B0C0D0E0F
   Nonce:           F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
   Associated data: 6173736F6369617465642064617461
   Message:         6D657373616765
   Padded A. data:  6173736F636961746564206461746101
   Padded Message:  6D65737361676501000000000000000
   CipherText:      8C49773E01CF46
   Tag:             9EB668AE9908201262D8F14682800D52
```

CHAPTER 4

Indices and tables

- genindex
- search