SpongeShaker

Release 1.1

November 22, 2014

Contents

This module implements Keccak-f1600 sponge permutation and high-level APIs for various modes of it, including SHA-3 hashes.

SHA-3 standard is not finalized, so actual output values are not stable yet. This implementation is up-to-date with Apr-2014 draft of FIPS-202. (Although it's unlikely that final SHA-3 changes hash parameters or padding again, instead they might add more modes.)

Features:

- Hashing (SHA3), PRNG, Stream cipher, AEAD cipher (SpongeWrap).
- Optimized-C implementation from Keccak reference code, with separate paths for 64- and 32-bit CPUs.
- Works with both Python 2.x and 3.x.

Todo:

- Sync with final SHA-3.
- Optimized ASM implementations.
- Other Keccak permutation sizes.
- Other sponge algorithms.
- Other sponge modes.

Links:

- Downloads
- Git repo

Documentation:

CHAPTER 1

spongeshaker API

Table Of Contents

1.1 `spongeshaker.sha3` - SHA-3 proposal

Proposed SHA3 algorithm selection.

Current values correspond to draft FIPS 202 (Apr 2014).

`spongeshaker.sha3.`**`sha3_224`**(*data=None*)
Proposed SHA3-224 by NIST (c=448).

Security level: 112/224 bits.

Returns `spongeshaker.hashing.SpongeHash` for SHA3-224.

`spongeshaker.sha3.`**`sha3_256`**(*data=None*)
Proposed SHA3-256 by NIST (c=512).

Security level: 128/256 bits.

Returns `spongeshaker.hashing.SpongeHash` for SHA3-256.

`spongeshaker.sha3.`**`sha3_384`**(*data=None*)
Proposed SHA3-384 by NIST (c=768).

Security level: 192/384 bits.

Returns `spongeshaker.hashing.SpongeHash` for SHA3-384.

`spongeshaker.sha3.`**`sha3_512`**(*data=None*)
Proposed SHA3-512 by NIST (c=1024).

Security level: 256/512 bits.

Returns spongeshaker.hashing.SpongeHash for SHA3-512.

spongeshaker.sha3.**shake128**(*data=None*, *digest_size=256*)
Proposed SHAKE128 hash by NIST (c=256).

Security level: 128 bits.

Parameters:

data initial data to hash.

digest_size Output size for .digest()/.hexdigest() when used as normal hash. Default: 256 bits.

Returns spongeshaker.hashing.SpongeHash for SHAKE128.

spongeshaker.sha3.**shake256**(*data=None*, *digest_size=512*)
Proposed SHAKE256 hash by NIST (c=512).

Security level: 256 bits.

Parameters:

data initial data to hash.

digest_size Output size for .digest()/.hexdigest() when used as normal hash. Default: 512 bits.

Returns spongeshaker.hashing.SpongeHash for SHAKE256.

1.2 spongeshaker.hashing - Sponge as hash

High-level APIs to Keccak1600 algorithm.

exception spongeshaker.hashing.**SpongeHashInvalidState**
Bases: exceptions.Exception

Extracting has started, cannot .update()/.digest().

class spongeshaker.hashing.**SpongeHash**(*capacity_bits*, *output_bits*, *data=None*, *name=None*, *sponge_class=None*, *padding=None*, *_sponge=None*, *_extracting=False*)
Bases: object

Generic hashlib compatible hash function API.

Initialize sponge instance with specified parameters.

Parameters:

capacity_bits number of bits for capacity.

digest_bits number of bits for digest output.

data initial data to hash.

name User-visible name for hash+parameters.

sponge_class Sponge implementation class that implements the spongeshaker.sponge.Sponge interface.

padding Start bytes for padding bytes to use, final bit is always added.

copy()
Create copy of current state.

update(*data*)
Update state with data.

Cannot be used after `extract()` is called.

digest()
Return final hash digest.

This follows the `hashlib` convention that state is not changed so `update()` can be called again to add more data to state.

hexdigest()
Return `digest()` value as hexadecimal string.

extract(*count*)
Extract data from hash state.

This function can be continued to be called to extract unlimited amount of bytes from state.

It *does* change the state, so `update()`, `digest()` and `hexdigest()` will throw error after `extract()` has been called.

1.3 spongeshaker.prng - Sponge as PRNG

Pseudo-random number generator API.

class spongeshaker.prng.**SpongePRNG**(*sponge*, *padding='x01'*)
Bases: `object`

Sponge as PRNG.

Initialize PRNG state with specified Keccak variant.

add_entropy(*data*)
Import new random data into state.

get_random_bytes(*nbytes*)
Return random bytes from state.

class spongeshaker.prng.**KeccakPRNG**(*capacity=512*, *padding='x01'*)
Bases: `spongeshaker.prng.SpongePRNG`

Keccak as PRNG.

1.4 spongeshaker.stream_cipher - Sponge as Stream cipher

Stream cipher with Keccak.

Stream cipher is basically PRNG, seeded with key, that generarates bytes that are XORed with data.

That means each message/stream *must* have unique stream, either by having unique key or IV. Otherwise the data can be trivially recovered.

class spongeshaker.stream_cipher.**SpongeStreamCipher**(*sponge*, *initial_data_pad='x01'*, *data_pad='x01'*)
Bases: `object`

Keccak Stream Cipher.

Example encryption:

```
state = KeccakStream(576)
state.add_initial_data(key)
ciphertext = state.encrypt(cleartext)
mac = state.final_digest(16)
```

Example decryption:

```
state = KeccakStream(576)
state.add_initial_data(key)
cleartext = state.decrypt(cleartext)
mac = state.final_digest(16)
```

Set up Keccak stream with given capacity (in bits) and padding.

add_initial_data(*data*)
Add initial data - key, iv, extra plaintext.

encrypt(*plaintext*)
Encrypt data.

Return plaintext XOR-ed with keystream.

decrypt(*ciphertext*)
Decrypt data.

Return ciphertext XOR-ed with keystream.

1.5 spongeshaker.spongewrap - Sponge as AEAD cipher

SpongeWrap - AEAD encryption with sponge.

http://sponge.noekeon.org/SpongeDuplex.pdf

class spongeshaker.spongewrap.**SpongeWrap**(*capacity=512*, *sponge_class=<type 'keccak.KeccakSponge'>*)
Bases: object

Authenticated encryption with sponge.

Each block is padded, padding includes "frame bit" before the standard 10*1 padding.

Frame bit = 0: next block will not be keystream. Frame bit = 1: next block will be keystream.

1.6 spongeshaker.sponge - Generic low-level sponge API

Low-level API for sponge implementations.

class spongeshaker.sponge.**Sponge**(*capacity*)
Bases: object

Generic sponge API.

Initialze sponge with given capacity (in bits).

absorb(*data*)
Add data to sponge by XOR-ing it into state.

squeeze(*nbytes*)
Extract given number of bytes from state.

squeeze_xor(*data*)
Return data XOR-ed with state.

encrypt(*data*)
Return data XOR-ed into state.

The state should already absorbed and permuted before encrypting starts. This means .absorb(key) + .pad() should be called.

decrypt(*enc_data*)
Return enc_data XOR-ed with state.

This is reverse of .encrypt() - it assumes enc_data was created by XOR-ing current state with cleartext. This function reverses it.

pad(*suffix*)
pad(suffix) - Add padding and permute state.

The suffix is added into state, then also final bit is flipped. So to get original simple 10*1 padding given in the Keccak SHA3 proposal, the suffix needs to be '01'.

If padding is suffix is empty, then final bit is not flipped, to support case when initial data for encryption is added without padding - which is bad style.

rewind()
Move internal position to start of state.

Useful for PRNG/duplex modes. In fact, it should not be touched at all in other modes.

forget()
Clear internal state, except capacity.

copy()
Return new instance with same state.

1.7 spongeshaker.keccak - Low-level sponge API for Keccak

Implements Sponge API for Keccak-f1600.

class spongeshaker.keccak.**KeccakSponge**
Bases: object

KeccakSponge(capacity) - Initializes Sponge object with given capacity in bits.

absorb()
absorb(data) - XOR data into state.

capacity
Sponge capacity in bits

copy()
copy() - Copy current state to new object.

decrypt()
decrypt(enc_data) - return enc_data XOR-ed with state.

This is reverse of .encrypt() - it assumes enc_data was created by XOR-ing current state with cleartext. This function reverses it.

encrypt()
encrypt(data) - return data XOR-ed into state.

The state should already absorbed and permuted before encrypting starts. This means .absorb(key) + .pad() should be called.

forget()
forget() - clear internal state, except capacity.

name
Sponge name

pad()
pad(suffix) - Add padding and permute state.

The suffix is added into state, then also final bit is flipped. So to get original simple 10*1 padding given in the Keccak SHA3 proposal, the suffix needs to be '\x01'.

If padding is suffix is empty, then final bit is not flipped, to support case when initial data for encryption is added without padding - which is bad style.

pos
Current position in bytes

rate
Sponge rate in bits

rbytes
Current position in bytes

rewind()
rewind() - move internal position to start of state.

Useful for PRNG/duplex modes. In fact, it should not be touched at all in other modes.

squeeze()
squeeze(nbytes) - extract given number of bytes from state.

squeeze_xor()
squeeze_xor(data) - return data XOR-ed with state.

CHAPTER 2

spongeshaker

2.1 Version 1.1

- Keccak now uses optimized implementation by default.
- Fix potential crash on 32-bit architectures due to wrong sizeof.
- Generic base classes for PRNG and StreamCipher.

2.2 Version 1.0

- Initial release.

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

S

A

C

D

E

F

G

H

K

N

P

R

S

U