pcr Documentation

Release 0.6.0

Stefano Palazzo

January 13, 2014

Contents

CHAPTER 1

AES

Advanced Encryption Standard - Block Cipher

class `pcr.aes.`**`AES`**

static `rotate`(*word*)
rotate a sequence of bytes eight bits to the left

static `xor`(*a*, *b*)
bitwise xor on equal length bytearrays

CHAPTER 2

CBC

Cipher Block Chaining Mode of Operation

class `pcr.cbc.`**`CBC`**(*BlockCipher*, *iv*)

`decrypt`(*data*, *key*)

`encrypt`(*data*, *key*)

static `xor`(*a*, *b*)

CHAPTER 3

Diffie-Hellman

Diffie-Hellman Key Exchange

class pcr.diffie_hellman.**DiffieHellman**(*prime*, *generator*, *rand_max*)

get_public_key()

get_shared_secret(*yb*)

CHAPTER 4

HOTP (Two-Factor Authentication)

Time OTP implementation for 2-factor authentication

pcr.hotp.**get_token**(*secret*, *i=None*)

pcr.hotp.**new_secret**()

CHAPTER 5

Maths

Various mathematical function used in public key cryptography

pcr.maths.**is_prime**(*n*, *k=64*)
Test whether n is prime using the probabilistic Miller-Rabin primality test. If n is composite, then this test will declare it to be probably prime with a probability of at most 4**-k.

To be on the safe side, a value of k=64 for integers up to 3072 bits is recommended (error probability = 2**-128). If the function is used for RSA or DSA, NIST recommends some values in FIPS PUB 186-3:

<http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf>

Do not use this function for small numbers.

pcr.maths.**get_prime**(*bits*, *k=64*)
Return a random prime up to a certain length

This function uses random.SystemRandom.

pcr.maths.**phi**(*n*, *p*, *q*)
Euler's totient function for n which can be written as pq

This is the number of k in the range 0 <= k <= n where gcd(n, k) is = 1 or, in other words, the number of integers k <= n that are relatively prime to n.

pcr.maths.**mult_inv**(*a*, *b*)
Calculate the multiplicative inverse a**-1 % b

This function works for n >= 5 where n is prime.

pcr.maths.**make_rsa_keys**(*bits=2048*, *e=65537*, *k=64*)
Create RSA key pair

Returns n, e, d, where (n, e) is the public key and (n, e, d) is the private key (and k is the number of rounds used in the Miller-Rabin primality test).

CHAPTER 6

PBKDF2

Password based key-derivation function - PBKDF2

`pcr.pbkdf2.`**`pbkdf2`**(*digestmod*, *password*, *salt*, *count*, *dk_length*)

PBKDF2, from PKCS #5 v2.0: http://tools.ietf.org/html/rfc2898

For proper usage, see NIST Special Publication 800-132: http://csrc.nist.gov/publications/PubsSPs.html

The arguments for this function are:

digestmod a crypographic hash constructor, such as hashlib.sha256 which will be used as an argument to the hmac function. Note that the performance difference between sha1 and sha256 is not very big. New applications should choose sha256 or better.

password The arbitrary-length password (passphrase) (bytes)

salt A bunch of random bytes, generated using a cryptographically strong random number generator (such as os.urandom()). NIST recommend the salt be _at **least**_ 128bits (16 bytes) long.

count The iteration count. Set this value as large as you can tolerate. NIST recommend that the absolute minimum value be 1000. However, it should generally be in the range of tens of thousands, or however many cause about a half-second delay to the user.

dk_length The lenght of the desired key in bytes. This doesn't need to be the same size as the hash functions digest size, but it makes sense to use a larger digest hash function if your key size is large.

CHAPTER 7

PKCS7

PKCS7 Padding for Block Cipher Modes

pcr.pkcs7.**pad**(*data*, *block_size*)

pcr.pkcs7.**unpad**(*data*)

pcr.pkcs7.**check_padding**(*data*, *block_size*)

CHAPTER 8

RC4

RC4 stream cipher

`pcr.rc4.`**`key_schedule`**`(`*key*`)`

`pcr.rc4.`**`key_stream`**`(`*s*`)`

CHAPTER 9

RFC 3526

Groups for Diffie-Hellman as defined by RFC 3526

To get access to the 2048 bit group, for example, type:

```
>>> prime, generator = rfc3526.groups[2048]
```

See http://tools.ietf.org/html/rfc3526 for notes on usage.

CHAPTER 10

XTEA

XTEA block cipher (32 rounds)

`pcr.xtea.`**`encrypt`**(*block*, *key*)

`pcr.xtea.`**`decrypt`**(*block*, *key*)

CHAPTER 11

Indices and tables

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

p