Applied Cryptography

$_{s}ize An integer value; the size of the digest produced by the hashing objects. You could also obtain this value by creating a sam$

```
>>> from Crypto.Hash import MD5
>>> m = MD5.new()
>>> m.update('abc')
>>> m.digest()
'\x90\x01P\x98<\xd2O\xb0\xd6\x96?}(\xe1\x7fr'
>>> m.hexdigest()
'900150983cd24fb0d6963f7d28e17f72'
```

$^{b}10"and"IoweBob$

```
>>> from Crypto.Cipher import DES
>>> obj=DES.new('abcdefgh', DES.MODE_ECB)
>>> plain="Guido van Rossum is a space alien."
>>> len(plain)
34
>>> obj.encrypt(plain)
Traceback (innermost last):
  File "<stdin>", line 1, in ?
ValueError: Strings for DES must be a multiple of 8 in length
>>> ciph=obj.encrypt(plain+'XXXXXX')
>>> ciph
'\021,\343Nq\214DY\337T\342pA\372\255\311s\210\363,\300j\330\250\312\347\342I\3215w\03561\303dgb/\00
>>> obj.decrypt(ciph)
'Guido van Rossum is a space alien.XXXXXX'
```

$_{C}BC or MODE_{C}FB, IV must be provided, and must be a string of the same length as the block size. Some algorithms suppo$
$_{s}ize An integer value; the size of the blocks encrypted by this module. Strings passed to the encrypt and decrypt functions m$
$_{s}ize An integer value; the size of the keys required by this module. If key_{s}ize is zero, then the algorithm accepts arbitrary -$
$_{s}ize An integer value equal to the size of the blocks encrypted by this object. Identical to the module variable of the same name$
$_{s}ize An integer value equal to the size of the keys used by this object. If key_{s}ize is zero, then the algorithm accepts arbitrary$
$.
$_{s}ize.
$_{s}ize. The output is a string object.

```
pow(blocksper, int(factor * number-of-blocks))
```

```
>>> from Crypto.Hash import MD5
>>> from Crypto.PublicKey import RSA
>>> RSAkey = RSA.generate(384, randfunc)   # This will take a while...
>>> hash = MD5.new(plaintext).digest()
>>> signature = RSAkey.sign(hash, "")
>>> signature   # Print what an RSA sig looks like--you don't really care.
('\021\317\313\336\264\315' ...,)
>>> RSAkey.verify(hash, signature)     # This sig will check out
1
>>> RSAkey.verify(hash[:-1], signature)# This sig will fail
0
```

$_{f}unc = None Generate a fresh public/private key pair. size is a algorithm - dependent size parameter, usually measure$
$don't
$_{f}unc is an optional function that will be called with a short string containing the key parameter currently being generated; i$
$_{b}lind Returns true if the algorithm is capable of blinding data; returns false otherwise.
$_{e}ncrypt Returns true if the algorithm is capable of encrypting and decrypting data; returns false otherwise. To test if a giv$
$_{s}ign Returns true if the algorithm is capable of signing data; returns false otherwise. To test if a given key object can signd$
$_{p}rivate Returns true if the key object contains the private key data, which will allow decrypting data and generating signat$
$de facto
n

```
for i in range(2, n):
    if (n%i)==0:
        print i, 'is a factor'
        break
```

$n
$_{b}ytes() method of a RandomPool object will serve the purpose nicely, as will the read() method of an opened file such as /dev$
$_{e}venttime, string Adds an event to the random pool. time should be set to the current system time, measured at the highest re$
$_{e}vent() method is determined, and the entropy of the data is guessed; the larger the time between calls, the better. The syste$
$_{e}vent() method, and decreased by the get_{b}ytes() method.
$_{b}ytes num Returns a string containing num bytes of random data, and decrements the amount of entropy available. It is no$
$_{e}vent().
$_{t}o_{e}nglish key Accepts a string of arbitrary data key, and returns a string containing uppercase English words separated by$
$_{t}o_{k}ey string Accepts string containing English words, and returns a string of binary data representing the key. Words mu$

```
#define MODULE_NAME MD2 /* Name of algorithm */
#define DIGEST_SIZE 16          /* Size of resulting digest in bytes */
```

$_{s}tate$:

```
typedef struct {
     ... whatever state variables you need ...
} hash_state;
```

$_{i}nit(hash_{s}tate * self); void hash_{u}pdate(hash_{s}tate * self, unsigned char * buffer, int length);$
$_{d}igest(hash_{s}tate * self); void hash_{c}opy(hash_{s}tate * source, hash_{s}tate * dest);$
$_{t}emplate.c" at the end of the file to include the actual implementation of the module.$

```
#define MODULE_NAME AES        /* Name of algorithm */
#define BLOCK_SIZE 16          /* Size of encryption block */
#define KEY_SIZE 0             /* Size of key in bytes (0 if not fixed size) */
```

$_{s}tate$: