Getting started

Python doesn't have very much in its standard library that deals with encryption. Instead, you get hashing libraries. We'll take a brief look at those in the chapter, but the primary focus will be on the following 3rd party packages: PyCrypto and cryptography. We will learn how to encrypt and decrypt strings with both of these libraries.

Hashing

If you need secure hashes or message digest algorithms, then Python's standard library has you covered in the **hashlib** module. It includes the FIPS secure hash algorithms SHA1, SHA224, SHA256, SHA384, and SHA512 as well as RSA's MD5 algorithm. Python also supports the adler32 and crc32 hash functions, but those are in the **zlib** module.

One of the most popular uses of hashes is storing the hash of a password instead of the password itself. Of course, the hash has to be a good one or it can be decrypted. Another popular use case for hashes is to hash a file and then send the file and its hash separately. Then the person receiving the file can run a hash on the file to see if it matches the hash that was sent. If it does, then that means no one has changed the file in transit.

Let's try creating an md5 hash:

```
import hashlib
md5 = hashlib.md5()
md5.update('Python rocks!')
#Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 3, in <module>
#    md5.update('Python rocks!')
#TypeError: Unicode-objects must be encoded before hashing
```

```
import hashlib
md5 = hashlib.md5()


md5.update(b'Python rocks!')
print (md5.digest())
#b'\x14\x82\xec\x1b#d\xf6N}\x16*+[\x16\xf4w'
```

Let's take a moment to break this down a bit. First off, we import **hashlib** and then we create an instance of an md5 HASH object. Next we add some text to the hash object and we get a traceback. It turns out that to use the md5 hash, you have to pass it a byte string instead of a regular string. So we try that and then call it's **digest** method to get our hash. If you prefer the hex digest, we can do that too:

```
import hashlib
md5 = hashlib.md5()

print (md5.hexdigest())
#'d41d8cd98f00b204e9800998ecf8427e'
```

There's actually a shortcut method of creating a hash, so we'll look at that next when we create our sha512 hash:

```
import hashlib
sha = hashlib.sha1(b'Hello Python').hexdigest()
print (sha)
#'422fbfbc67fe17c86642c5eaaa48f8b670cbed1b'
```

As you can see, we can create our hash instance and call its digest method at the same time. Then we print out the hash to see what it is. I chose to use the sha1 hash as it has a nice short hash that will fit the page better. But it's also less secure, so feel free to try one of the others.