OliCyber.IT 2023 - Approfondimenti
PyCryptodome basics

Contenuti

1 PyCryptodome basics

PyCryptodome[1] è una API[2] Python di primitive crittografiche: fondamentale per poter scrivere degli script senza dover implementare tutti gli algoritmi da zero.
Non è l'unica ad adempiere a questo compito, ma per immediatezza e facilità di utilizzo è a mio parere la migliore da imparare a usare nel momento in cui si comincia a risolvere qualche challenge non banale di crittografia - che richieda di utilizzare un qualche cifrario a blocchi come AES, ad esempio.
Se mastichi un po' d'inglese, al link fornito c'è tutta la documentazione: suggerisco caldamente di darci un'occhiata, eventualmente con l'aiuto di Google Translate. Altrimenti, qui cercherò di fare un riassunto (tradotto) delle informazioni necessarie per poter approcciare le challenge presenti in piattaforma.

1.1 Installazione

È sufficiente dare il comando:

```
$ pip install pycryptodome
```

da terminale. Fai attenzione a non aver installato accidentalmente (attraverso qualche dipendenza automatica) anche la libreria PyCrypto: interferiranno l'una con l'altra.
PyCrypto è una vecchia libreria utilizzata per questi scopi; PyCryptodome è una sua diramazione (quasi del tutto retrocompatibile) che porta notevoli miglioramenti.
Puoi controllare questa eventualità con il comando:

```
$ pip freeze | grep -i pycrypto
```

In caso puoi disinstallare PyCrypto con:

```
$ pip uninstall pycrypto
```

1.2 Struttura della libreria

La libreria è organizzata in pacchetti, ciascuno designato per racchiudere funzioni e classi atte a risolvere specifici problemi.
In Python è sufficiente importare il modulo desiderato, per poi accedere agli attributi/metodi con il consueto ".":

```
>>> from Crypto.Cipher import AES
>>> AES.new(key, AES.MODE_ECB)
```

In alternativa è possibile importare solo le funzioni desiderate (una per una oppure tutte quelle disponibili, utilizzando la wildcard "*"):

```
>>> from Crypto.Random import get_random_bytes
>>> from Crypto.Random.random import getrandbits
>>> getrandbits(64)
11380094873065783537
>>> from Crypto.Util.Padding import *
>>> pad(b'ciao', 16)
b'ciao\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c'
```

[1] https://pycryptodome.readthedocs.io/en/latest/src/introduction.html
[2] https://it.wikipedia.org/wiki/Application_programming_interface

Pacchetto	Modulo
Crypto.Cipher	ChaCha20 Salsa20 AES DES/DES3
Crypto.Hash	MD5 SHA1 SHA{224/256/384/512} SHA3_{224/256/384/512} HMAC CMAC
Crypto.PublicKey	RSA DSA ElGamal
Crypto.Util	Padding strxor number
Crypto.Random	get_random_bytes(N) random.getrandbits(N)

Vediamo ora una breve descrizione di ciascuna delle API menzionate:

- `Crypto.Cipher` si occupa della gestione degli algoritmi di cifratura/decifrazione per diversi schemi (simmetrici o asimmetrici). Si crea un'istanza dell'oggetto chiamando la funzione `new()`, che richiede diversi parametri a seconda del modulo (chiave, modalità di operazione, nonce, vettore di inizializzazione eccetera). Per cifrare/decifrare basta invocare i metodi `encrypt()`/`decrypt()` dell'oggetto creato, passando come argomento il testo in chiaro/testo cifrato (`bytes`, `bytearray` (, `memoryview`)).

- `Crypto.Hash` è dedicato all'implementazione di funzioni di hash e message authentication codes. Anche qui si crea un'istanza dell'oggetto con la funzione `new()`. Si aggiorna dunque lo stato dell'oggetto con il metodo `update()` (anche qui l'input dev'essere fornito come `bytes`, `bytearray`). Per produrre l'output (digest) si invocano i metodi `digest()`, `hexdigest()` a seconda che si vogliano ottenere `bytes` o `hex`.

- `Crypto.PublicKey` serve per la creazione, manipolazione, importazione ed esportazione di chiavi pubbliche. Solitamente in ambito CTF viene utilizzato per la lettura e codifica/decodifica delle chiavi nei formati standard, di solito molto fastidiosi da gestire a mano (X.509[3] / PKCS#1[4] / PKCS#8[5] binari, DER[6] o PEM[7]). Ciononostante, le chiavi importate o create possono essere utilizzate dagli algoritmi a chiave pubblica disponibili nel pacchetto `Crypto.Cipher`.
  In questo caso un oggetto "chiave" può essere istanziato in quattro modi, vediamo i tre principali:
  - con la funzione `generate()` la chiave è generata casualmente
  - `import_key()` permette di caricare una chiave dalla memoria (nei formati menzionati poc'anzi)
  - `construct()` consente di costruire una chiave da un insieme di componenti (si pensi al caso RSA, in cui è necessario fornire e, N per la chiave pubblica, mentre per la privata corrispondete bisogna aggiungere anche d).

- `Crypto.Util` è un pacchetto che gestisce vari moduli miscellanei. Di particolare rilievo sono:
  - `Padding` (l'utilizzo del modulo si intuisce dal nome; sono disponibili gli algoritmi PKCS#7, iso7816, x923)
  - `strxor` (anche qui abbastanza immediato, serve a produrre lo xor tra due oggetti `bytes`/`bytearray`)
  - `number`, nello specifico le funzioni per la generazione di numeri primi e controllo di primalità di un numero dato `getPrime()`, `isPrime()`

[3] https://it.wikipedia.org/wiki/X.509
[4] https://www.ietf.org/rfc/rfc3447.txt
[5] https://www.ietf.org/rfc/rfc5208.txt
[6] https://en.wikipedia.org/wiki/X.690
[7] https://en.wikipedia.org/wiki/Privacy-Enhanced_Mail

- infine `Crypto.Random` offre una funzione (`get_random_bytes()`) e un modulo (`random`) per la generazione casuale sicura di numeri (di una determinata lunghezza in bytes/bit). È praticamente un'estensione dell'usuale modulo standard `random`, con la **grossa** distinzione che in questo caso non viene utilizzato un PRNG, ma `os.urandom()`.