Encryptedly Yours : Python & Cryptography

SciPy India 2011

Nivedita Datta
Student, M.Sc engg.
IISc, SERC dept.

Outline

- What and why of cryptography?
- Basic terminologies
- Classification of cryptographic algorithms
- Hashing
- Introduction to PyCrypto
- Installation steps
- Examples
- Conclusion
- References

What and why of cryptography?

- Applied branch of mathematics
- Science and study of secret writing
- An excerpt from “The adventure of The Dancing men” by Arthur Canon Doyle

Courtesy : http://www.eastoftheweb.com/short-stories/UBooks/AdveDanc.shtml

- It is used to give :
  - Confidentiality
  - Integrity
  - Authentication
  - Authorization
  - Non-repudiation

Speak Like a Crypto Geek

- ***Plaintext*** – A message in its readable format
- ***Ciphertext*** – Message altered such that it is unreadable by anyone except the intended recipients
- ***Key*** – Sequence that controls the operation and behavior of the cryptographic algorithm
- ***Initialization Vector –*** Random string used while encrypting to avoid any pattern creation.
- ***Cryptosystem*** – Algorithm, key and key management functions together to perform cryptographic operations

Classifications of Cryptography

- Stream-based Ciphers
  - Encrypts one byte at a time
  - Mixes plaintext with key stream
- Block Ciphers
  - Encrypts a block of plaintext at a time
  - Generally used in chain mode to hide repeated plaintext blocks.
- Symmetric
  - Same key for encryption and decryption
- ***Asymmetric***
  - Mathematically related key pairs for encryption and decryption (public key & private key)

Hash Functions

- Inputs a data block and outputs a fixed-size string known as **hash value**.
- Four main or significant properties:
  - Easy to compute
  - Should be a one way function i.e. Infeasible to generate message from given hash
  - Infeasible to modify a message without changing the hash
  - Should be collision-free i.e. no two different message should give same hash value

Pycrypto : An introduction

- A module for writing python programs with cryptographic functions.
- Simple, consistent interface for similar classes of algorithm
- Some modules written in C, some in python
- OS used : Windows 7, 32 bits
- Software Version used : PyCrypto 2.3, Python 2.6.6
- Compatible with Python 2.x, not Python 3.x
- Latest version : 2.4.1

Installing PyCrypto

- Pre-requisite : Python
- On Windows :
  - http://www.voidspace.org.uk/python/modules.shtml#pycrypto
  - Download the PyCrypto version compatible with installed version of Python
  - Extract and run .msi file.

- On Ubuntu :
  - Type the following on the terminal :

    sudo apt-get install python python-crypto

  Note : System must be connected to the internet

AES (Advanced Encryption Standard)

- Private key block cipher
- Data block size : 128 bits
- Key length : 128, 192 & 256 bits
- Flavors referred to as AES-128, AES-192 & AES-256
- Number of rounds depends on key size.
- Syntax :

  AES.new(key, [mode],[IV])
- Modes :

  MODE_EBC = 1

  MODE_CBC = 2

  MODE_CBF = 3

  MODE_PGP = 4

  MODE_OBF = 5

  MODE_CTR = 6

AES : Example

```
from Crypto.Cipher import AES
import hashlib
import string

password = 'kitty‘

#digest() function outputs 16 bytes value
key = hashlib.sha256(password).digest()

# mode optional, default MODE_EBC or 1
# IV optional; 16 bytes long
encryptor = AES.new(key, 2)

# the block size for AES : 16, 24, or 32
BLOCK_SIZE = 32

text = ‘This is PyCrypto demo for AES’
PADDING = ‘ ’
```

```
# Padding with extra characters to make
# the text length multiple of BLOCK_SIZE
text = text + (BLOCK_SIZE - len(text) %
BLOCK_SIZE) * PADDING

ciphertext = encryptor.encrypt(text)
print ciphertext

decryptor = AES.new(key, 2)
plain = decryptor.decrypt(ciphertext)
print plain
```

ARC4 : Alleged RC4

- Private key stream cipher
- Data block size : 1 byte
- Key size : Varies between 1 to 256 bit
- Normally uses 64 bit or 128 bit key size
- Cryptographically very strong and easy to implement
- Data stream simply XORed with the generated key sequence

- Syntax:

  ARC4.new(…)

ARC4 : Example

```
from Crypto.Cipher import ARC4

key = '01234567'
encryptor = ARC4.new(key)
decryptor = ARC4.new(key)

text = ‘This is PyCrypto demo for ARC4’

#Encrypting Data
Cipher_text = encryptor.encrypt(text)
print cipher_text

#Decrypting Data
plain_text = decryptor.decrypt(cipher_text)
print plain_text
```

RSA : Rivest-Shamir-Adleman

- Public key cryptographic algorithm
- Named after Ron Rivest, Adi Shamir and Len Adleman
- Used for public key encryption & digital signatures.
- Security is based on the difficulty of factoring large integers.
- Key length : any length, minimum recommendation is 1024 bits.
- Syntax:

```
RSA.generate(size, randfunc, progress_func=None)
```

RSA : Example

```
from Crypto.Hash import MD5
from Crypto.PublicKey import RSA
from Crypto import Random

def display_progress(prog):
    print "generated", prog,

plaintext = "This is PyCrypto demo for
RSA"

key = RSA.generate(1024,
Random.new().read, display_progress)
hash = MD5.new(plaintext).digest()
signature = key.sign(hash,
Random.random.getrandbits(128))

# this public key is shared with the
#recipient or published somewhere
pub = key.publickey()

#Printing the value of public key
print "The public key is : ", pub, "\n"

#Algorithm name
print "Algorithm used is : ",
(RSA.__name__),"\n"

# now plaintext is sent to the recipient
ciphertext =
pub.encrypt(plaintext,key.p*key.q)
print "sent... ", ciphertext,"\n"

# the recipient can then decrypt the
# message and retrieve the original text
pt = key.decrypt(ciphertext)
print "received." ,pt,"\n"
```

RSA : Example (contd…)

```
#Sender sends plaintext & signature
# the recipient calculates the hash of
#message & confirms the message was
#not altered

print "The message is genuine:",
pub.verify(MD5.new(plaintext).digest(),sig
nature)
```

Hashing Algorithms : Example

```
from Crypto.Hash import HMAC
from Crypto.Hash import MD5, MD4, MD2,
import hashlib

text = ' This is PyCrypto demo for Hashing'

# key length : 32 bytes
hashkey = MD4.new(text).hexdigest()
print 'MD4 : '+hashkey

# key length : 32 bytes
hashkey = MD2.new(text).hexdigest()
print 'MD2 : '+hashkey

# key length : 64 bytes
hashkey = SHA256.new(text).hexdigest()
print 'SHA256 : '+hashkey
```

```
# to verify correctness of password
from Crypto.Hash import MD5

def chk_pswd(user_pswd,pswd_hash):
    return MD5.new(user_pswd).hexdigest()
==pswd_hash
```

Finally…

- Used in commercial products like Google App Engine
- Can be used in any application dealing with sensitive data
- Can be extended to include code for other cryptographic & hashing algorithms
- PyCrypto functions can also be called from program in other language.

- Want to contribute?
  - Join the PyCrypto mailing list in www.pycrypto.org

References

- http://docs.python.org/tutorial/index.html
- http://packages.python.org/pycrypto/
- http://www.pycrypto.org/
- http://www.voidspace.org.uk/python/modules.shtml#pycrypto
- Applied cryptography, second edition by Bruce Schneier

Thank You for your attention…!!