§7.6-7.8: Applications: Cæsar Cipher, Pseudo-random Numbers

17 Oct 2008
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Cryptography example

- Cæsar substitution cipher:
  - Key: e.g., QAZXSWEDCVFRTGBNHYUJMKIOLP
  - Cleartext: input text to encrypt
  - Ciphertext: output encrypted text
  - Encoding: replace each letter in source with corresponding letter from code key
  - Decoding: same, using the decode key
- ROT13 was an example of a substitution cipher
  - Key: NOPQRSTUVWXYZABCDEFGHIJKLM

Write a Substitution cipher library

- Design a public interface for the library?

```
def encode (src, key):
    """Encode the source string using the given
      codestring.
    Returns the encoded string.
    pre: src must be a string;
    key must be a permutation of the 26 letters."""
def decode (src, key):
    """Decode the source string using the given
      codestring.
    Returns the decoded string.
    pre: src must be a string;
    key must be a permutation of the 26 letters."""
```

Internal helper functions

- In the implementation it is handy to have some helper functions for internal use:

```
def isalpha (ch):
    """Return true if ch is a letter."""
def alpha_pos (ch):
    """Return index of a letter in the range 0 .. 25"""
def decode_key (enckey):
    """Create a decode key from an encoding key"""
```

- How to implement these?
  - isalpha() is built-in: ch.isalpha()

Implementing Substitution library

- Main function to encode strings:

```
def encode(src, key):
    """Encode the source string using the given
      codestring.
    Returns the encoded string.
    pre: src must be a string;
    key must be a permutation of the 26 letters.
    """
    dst = ""
    for ch in src:
        if ch.isalpha():
            dst += key[alpha_pos(ch)]
        else:
            dst += ch
    return dst
```

Implementing decode()

- Decoding is just encoding using a reverse key:

```
def decode (src, key):
    """Decode the source string using the given
      codestring.
    Returns the decoded string.
    pre: src must be a string;
    key must be a permutation of the 26 letters.
    """
    return encode(src, decode_key(key))
```

- **Library: http://twu.seanho.com/python/substitution.py**
- **Testbed: http://twu.seanho.com/python/caesartest.py**

Application: Random numbers

- A random number (from a uniform distribution) is chosen such that every number within the range is equally likely to be chosen:
  - Uniform distribution on [0..1]
- Making things truly random (high entropy) is very difficult!
  - Hardware random-number generators:
    - Measure radioactive decay of isotopes
    - Brownian motion of particles in a suspension (air)
  - Software pseudo-random number generators

Pseudo-random number generator

- A pseudo-random number generator applies some math operations to the last number generated to get the next number
  - Start with a seed number
  - Hopefully it's "random enough"
  - But really it's completely deterministic:
    - If we start again with the same seed, we'll always get the same sequence of "random" numbers
- e.g., seed=0.10: generates
  - 0.72, 0.23, 0.19, 0.93, 0.54, 0.77, 0.11, ...

DEF: pseudo-random num library

- We only need one public procedure: Random()

```
def random ():
    """Returns a random float between 0 and 1."""

def init_seed (x):
    """Initialize the number generator seed."""
```

- init_seed provides a way for the user to manually set the seed.

IMP: pseudo-random num library

```
"""Pseudo-random number generator.


Sean Ho
CMPT14x example 2006.
"""
from math import exp, log, pi


seed = 0            # persistent across calls to random()
def init_seed (x):
    """Initialize the number generator seed.
    Accessor (set) function for seed."""
    global seed             # access global variable
    seed = x
```

IMP: pseudorandom.py, cont.

```
def random ():
    """Returns a random float between 0 and 1."""
    global seed           # access global variable

    # Try to scramble up seed as much as possible
    seed = seed + pi
    seed = exp (7.0 * log (seed))

    # Only keep the fractional part, in range 0..1
    seed = seed – int (seed)
    return seed
```

Online test of PseudoRandom

- (demo in Python of PseudoRandomTest)
- Library: http://twu.seanho.com/python/pseudorandom.py

- Evaluating "randomness":
  - Graphical evaluations: plot points (x,y) where both coordinates are from Random()
  - Check for dense spots, sparse spots in 1x1 square
  - Python has graphics libraries, but that's beyond the scope of this class

Python's own pseudorandom

- Python has a built-in pseudorandom generator:

```
from random import random
random()
seed()
```

  - Random float in interval [0.0, 1.0)
- Histogram to evaluate randomness
  - Split up interval [0.0, 1.0) into equal-size bins
  - Generate a list of random numbers
  - Count how many numbers fall in each bin