FINE 434: FinTech

Lecture 10

Professor Fahad Saleh

McGill University - Desautels

Background

Python is an object-oriented programming language, which means it manipulates programming constructs called **objects**. You can think of an object as a single data structure that contains data as well as functions; the functions of an object are called its **methods**.

Syntax

```
from Crypto.Util import number
import hashlib

class FINE434DSA(object):

    def __init__(self, bitsp = 512, bitsq = 510):
        # Your Code Here

    def S(self, message):
        # Your Code Here

    def V(self, publicKey, signature, message):
        # Your Code Here
```

from ... import ...

```
from Crypto.Util import number
import hashlib
```

We use “from ... import ...” to import a specific resource from a specific package or module. In this case, we import “number” from “Crypto.Util” which we subsequently use.

init(...)

This function is required for classes, and it's used to initialize the objects it creates. _init_(...) always takes at least one argument, self, that refers to the object being created. You can think of _init_(...) as the function that "boots up" each object the class creates.

```
from Crypto.Util import number
import hashlib

class FINE434DSA(object):

    def __init__(self, bitsp = 512, bitsq = 510):
```

self

This is a Python convention; there's nothing magic about the word self. However, it's overwhelmingly common to use self as the first parameter in _init_(...), so you should do this so that other people will understand your code.

Python will use the first parameter that _init_(...) receives to refer to the object being created; this is why it's often called self, since this parameter gives the object being created its identity.

```
from Crypto.Util import number
import hashlib

class FINE434DSA(object):

    def __init__(self, bitsp = 512, bitsq = 510):
```

Getting Started

```
from Crypto.Util import number
import hashlib

class FINE434DSA(object):

    def __init__(self, bitsp = 512, bitsq = 510):
        p = number.getPrime(bitsp)
        q = number.getPrime(bitsq)
        self.N = p*q
```

Getting Started

```
from Crypto.Util import number
import hashlib

class FINE434DSA(object):

    def __init__(self, bitsp = 512, bitsq = 510):
        p = number.getPrime(bitsp)
        q = number.getPrime(bitsq)
        self.N = p*q
```

What's missing?

Getting Started II

```
from Crypto.Util import number
import hashlib

class FINE434DSA(object):

    def __init__(self, bitsp = 512, bitsq = 510):
        p = number.getPrime(bitsp)
        q = number.getPrime(bitsq)
        self.N = p*q
        self.e = number.getPrime(bitsp+bitsq-1)
        while self.e < max(p,q) or self.e > self.N:
            self.e = number.getPrime(bitsp+bitsq-1)
```

Getting Started II

```
from Crypto.Util import number
import hashlib

class FINE434DSA(object):

    def __init__(self, bitsp = 512, bitsq = 510):
        p = number.getPrime(bitsp)
        q = number.getPrime(bitsq)
        self.N = p*q
        self.e = number.getPrime(bitsp+bitsq-1)
        while self.e < max(p,q) or self.e > self.N:
            self.e = number.getPrime(bitsp+bitsq-1)
```

What else?

Your Turn

```python
from Crypto.Util import number
import hashlib

class FINE434DSA(object):

    def __init__(self, bitsp = 512, bitsq = 510):
        p = number.getPrime(bitsp)
        q = number.getPrime(bitsq)
        self.N = p*q
        self.e = number.getPrime(bitsp+bitsq-1)
        while self.e < max(p,q) or self.e > self.N:
            self.e = number.getPrime(bitsp+bitsq-1)
        # HW: assign a valid value for self.d
```

Instantiating an Object

```
myKey = FINE434DSA()
myOtherKey = FINE434DSA(10,12)
```

We instantiate an object by using the class name and then arguments pertaining to the _init_(...) declaration. We pass nothing for the first argument (self).

Dot Notation

```
myOtherKey = FINE434DSA(10,12)
print(myOtherKey.N)
print(myOtherKey.e)
```

We access data from our object by using dot notation. Specifically, to call a piece of data called y for an object named x, we write x.y.

Methods

When a class has its own functions, those functions are called methods. Classes may have as many functions as desired.

```
    def S(self, message):
        messagehash = hashlib.sha256(message.encode())
        signature = (int(messagehash.hexdigest(),16) ** self.d) % self.N
        return(signature)

    def V(self, e, N, signature, message):
        messagehash = hashlib.sha256(message.encode())
        return((signature ** e) % N == int(messagehash.hexdigest(),16) % N)
```

Digression: Tuple

RSA public keys, (e, N), and private keys, (d, N), are ordered pairs. Python has a type for such objects known as a "tuple."

```
myKey = FINE434DSA(6, 8)
myPublicKey = (myKey.e,myKey.N)
print("Public Key : "+str(myPublicKey))
print("e : "+str(myPublicKey[0]))
print("N : "+str(myPublicKey[1]))
```

```
Public Key : (4409, 8843)
e : 4409
N : 8843
```

Putting It Together

```
myKey = FINE434DSA(6, 8)
OtherKey = FINE434DSA(6, 8)
m = "I want an A!"
s = myKey.S(m)
s2 = OtherKey.S(m)
s1v = myKey.V(myKey.e,myKey.N,s,m)
s2v = myKey.V(myKey.e,myKey.N,s2,m)
print("Valid Signature: "+str(s1v))
print("Invalid Signature: "+str(s2v))
```

```
Valid Signature: True
Invalid Signature: False
```