PGP with ElGamal and Elliptic curves

Andrew Kwong

- What is PGP?
  - Pretty Good Privacy
  - Encryption software used for public key encryption
- What is public key cryptography?
Elliptic Curve Cryptography

- Pros:
  - Shorter key length. 256 bits vs. 3072 bits
  - Keys generated very quickly

- Cons:
  - RSA encrypts/decrypts more quickly
  - RSA is much less confusing
  - ECC is less resilient to quantum computing attack
NIST P-256

- Recommended by NSA through its Suite B
- Approved for encrypting documents classified as SECRET, though 384 bit key is required for TOP SECRET
- \( y^2 = x^3 + bx + c \pmod{p} \)

\[
p = 115792089210356248762697446949407573530086143415290314195533631308867097853951 \\
b = 115792089210356248762697446949407573530086143415290314195533631308867097853948 \\
c = 41058363725152142129326129780047268409114441015993725554835256314039467401291 \\
a_x = 48439561293906451759052585252797914202762949526041747995844080717082404635286 \\
a_y = 36134250956749795798585127919587881956611106672985015071877198253568414405109 \]
Python Review

- simplifies dealing with arbitrary precision integers
- Supports both imperative and functional programming

```
for x in xrange(len(plain_text)):
    plain_text[x] = ''.join(map(lambda x: chr(x), plain_text[x]))
```

- SLOW!
- Speed is very important for cryptographic functions
- Dynamically typed
- Can't curry functions like Haskell, have to use closure
Haskell Review

- Good for
- more clear what is happening (sometimes)
- Very concise code
- Type safety helps catch bugs early; statically typed
- Hopefully faster

```
encode::[String]->[Int]
encode x=map (foldl \(a b->1000*a+b\) 0) enc_blocks
    where enc_blocks = map (map ord x)

let blocks=splitEvery 4 encrypt_file
encoded= encode blocks
```

```
if encrypt_flag==True:
    #parse file
    f_encrypt_name=open(encrypt_name,'rb')
    points=[]
    i=0
    while True:
        c=f_encrypt_name.read(1)
        if not c:
            if i!=0:
                #flush buffer
                blocks.append(block)
            break
            block.append(c);
        i=i+1;
        if i==block_length:
            blocks.append(block)
            i=0;
            block=[]
    for a in blocks:
        a[0]=map(ord,a[0:]);
    a[1]=map(lambda x: int(binascii.hexlify(x), 16),a[0:])
    for a in blocks:
        val=reduce(lambda x,y: 1000*x+y,a,0)
        points.append(val)
```
Challenges

- Reviewing number theory/crypto knowledge
- Theory vs application
- Corner cases
- Dealing efficiently with huge numbers
  - Computing quadratic residues, modular exponentiation, modular inverses
  -skirts the code printed in the image.
  - Tempted to use gmopy