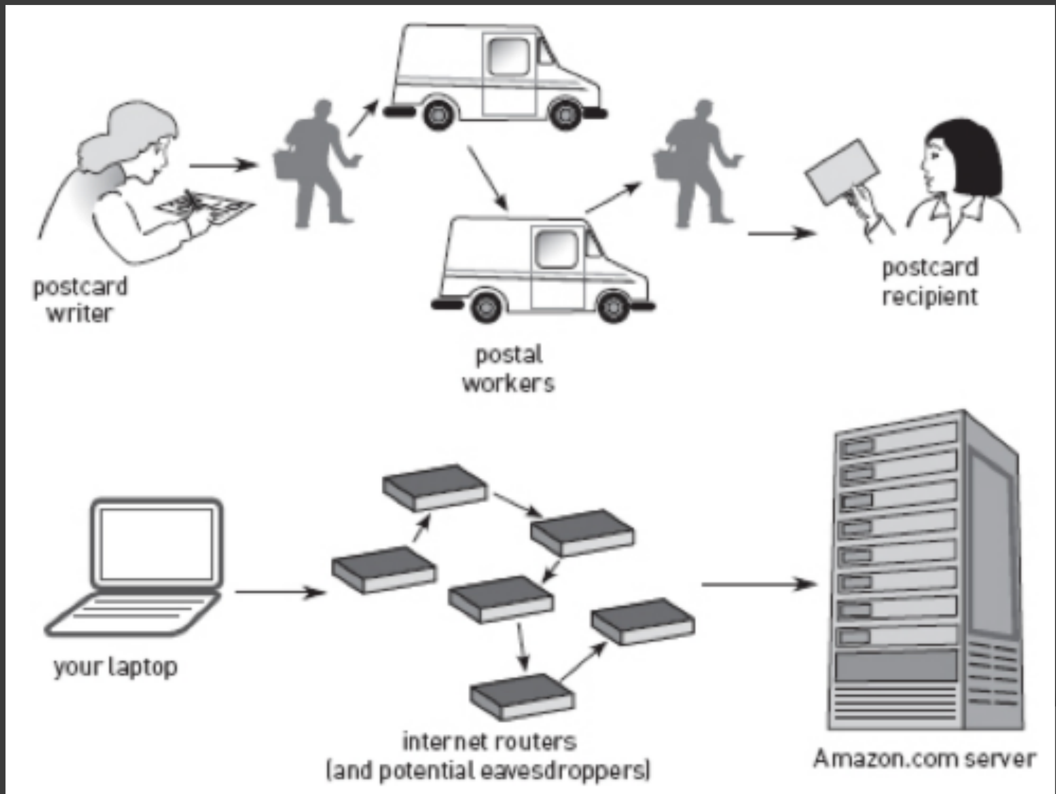
Damon Aitken & Yuxin Wu

ICS FINAL PROJECT: ENCRYPTED MESSAGING

Why is Encryption Necessary?

- Internet makes it easy to eavesdrop on what computers are saying



RSA Algorithm

- Devised in 1978 by (R)ivest, (S)hamir, and (A)dleman
- Based on earlier ideas of Diffie and Hellman
- Asymmetric algorithm
  - 2 keys – public and private
- Uses prime numbers and co-primes
  - Numbers are co-primes if the only factor that they share is 1

Why is this Algorithm Difficult to Break?

- Must know the private key
- Factoring integers is difficult
  - Many different algorithms try
  - GNFS is the best one so far

$$O(\exp\sqrt[3]{\frac{64}{9}b\left(\log b\right)^{2}})$$

- Especially difficult to factor large prime numbers
  - Nature of coprimes

Flow

1. Choose two random prime numbers
2. Calculate product to get <u>n</u> (shared number)
3. Calculate totient (product of each number subtracted by 1)
4. Find number <u>e</u> which is co-prime to the totient
5. Use (1 + totient)/e to get number <u>d</u>
6. Public key = (e, n)
7. Private key = (d, n)

Let’s find d!

- We need to solve the equation to find the solution. But this equation may have infinite solutions. Luckily we only need one of them.

```
def get_xy(self, a,b):
    for i in range(0,10000000):
        d = (1 + (i * b))/a
        if int(d) == d:
            return int(d)
```

Let’s find e !

- Use while loop to run the e generator until it finds a co-prime with n
- The gcd of co-primes is 1

```
def get_encryption(self,n):
    pd = True
    while pd == True:
        self.e = random.randint(4,n)
        if math.gcd(n, self.e) == 1:
            pd = False
    return self.e
```

Encrypting Public & Private Keys

```python
def get_pp(self):
    prime_tuple = self.prime_number(self.prime_list())
    for i in prime_tuple:
        number1 = prime_tuple[0]
        number2 = prime_tuple[1]
    n = self.get_n(number1,number2)
    phi = self.get_phi(number1,number2)
    e = self.get_encryption(phi)
    d = self.get_xy(e,phi)
    private = self.private_key(d,n)
    public = self.public_key(e,n)
    return private, public
```

- Generate e,d,n from two random prime numbers
- Assign to public and private keys

Encrypt & Decrypt Messages

```
def encrypt(self, pk, plaintext):
    #Unpack the key into it's components
    key, n = pk
    #Convert each letter in the plaintext to numbers based on
    cipher = [(ord(char) ** key) % n for char in plaintext]
    #Return the array of bytes
    return cipher
```

- Separate key
- Convert each letter to a random number based on key

```
def decrypt(self, pk, ciphertext):
    #Unpack the key into its components
    key, n = pk
    #Generate the plaintext based on the ciphertext and key us
    plain = [chr((char ** key) % n) for char in ciphertext]
    #Return the array of bytes as a string
    return ''.join(plain)
```

- Separate key
- Use the key to unscramble string

Client State Machine Setup

- Add self.pd in _init_ function
  - Keeps track of whether or not message has been encrypted before
- Empty () for private, public and peer_key
- Incorporate key generation functions into ClientSM class
- Encryption handled on client side so server doesn’t see

Modifying Chatting State

- Encrypt message function

```
elif self.state == S_CHATTING:
    if self.pd == False:
        self.private, self.public = self.get_pp()
        mysend(self.s, M_KEY + str(self.public[0]) + "#" + str(self.public[1]))
        self.pd = True

    if len(my_msg) > 0: # my stuff going out
        encrypt_msg = self.encrypt(self.peer_key, "[" + self.me + "]" + my_msg)
        send = ''
        for i in encrypt_msg:
            send += "#" + str(i)
        mysend(self.s, M_EXCHANGE + "[" + self.me + "]" + (send))

        if my_msg == 'bye':
            self.disconnect()
            self.state = S_LOGGEDIN
            self.peer = ''
            self.pd = False

    if len(peer_msg) > 0:    # peer's stuff, coming in
        if peer_code == M_CONNECT:
            self.out_msg += "(" + peer_msg + " joined)\n"
            self.pd = False    #to account for new connection

        if peer_code == M_KEY:
            peer_key1 = (peer_msg).split("#")
            self.peer_key = (int(peer_key1[0]), int(peer_key1[1]))

        else:
            jj = peer_msg.split("#")[1:]
            decrypt_msg = self.decrypt(self.private, jj)
            self.out_msg += decrypt_msg
```

Improvements

- Add functionality for multiple persons
- RSA can be breakable in the future
- Need to constantly stay ahead of breaking algorithms

Sources

- UT Texas. RSA Algorithm Example. https://www.cs.utexas.edu/~mitra/honors/soln.html
- MacCormick, John. Nine Algorithms That Changed the Future: The Ingenious Ideas That Drive Today's Computers. Princeton: Princeton University Press, 2012.
- Rivest, R.; A. Shamir; L. Adleman (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM 21 (2): 120–126. doi:10.1145/359340.359342.