Data Structures & Algorithms 2

Cryptography

Lecturer: Dr. Hadi Tabatabaee
Materials: Dr. Phil Maguire
Maynooth University
Online at http://moodle.maynoothuniversity.ie

Overview

Aims

- A review of data structures and algorithms in Cryptography

Learning outcomes: You should be able to…

- Use simple cryptography (e.g., Caesar cipher and Vignière Cipher)
- Use public key and private key to encrypt and decrypt
- Use Big numbers in encryption

Cryptography

- Cryptography has had a major impact on wars throughout history
- Communication between army battalions is vital for carrying out military operations
- But if a message gets intercepted then the consequences can be disastrous
- In ancient times people used very simple transposition and substitution ciphers

Cryptography

- The idea of cryptography is to ensure message confidentiality
- Messages should be convertible to an incomprehensible form to make them unreadable to eavesdroppers or interceptors
- The person receiving the message should then be able to convert the message back to its original form
- Caesar cipher: move every letter up a few positions
  - h e l l o → k h o o r

Vignière Cipher

- This is a form of polyalphabetic Caesar substitution, first described in 1553 - for 300 years it resisted all attempts to break it
- Instead of shifting every character up a set amount, you shift them differing amounts based on a repeated secret word

Plaintext:	ATTACKATDAWN
Key:	LEMONLEMONLE
Ciphertext:	LXFOPVEFRNHR

- In 1863 Friedrich Kasiski was the first to publish a successful general attack, which takes advantage of the fact that repeated words may, by chance, sometimes be encrypted using the same key letters, leading to repeated groups in the ciphertext

Key:	ABCDABCDABCDABCDABCDABCDABCD
Plaintext:	*CRYPTO*ISSHORTFOR*CRYPTO*GRAPHY
Ciphertext:	*CSASTP*KVSIQUTGQU*CSASTP*IUAQJB

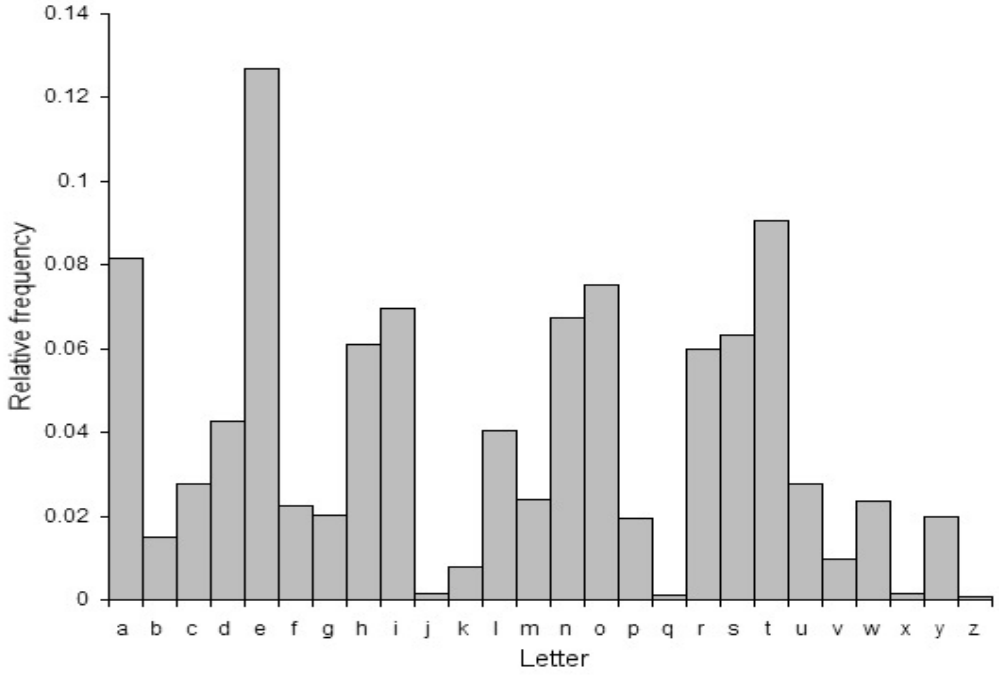
Cryptography

- The problem with ciphertexts is that they always betray statistical information about the plaintext
  - ‘e’ is the most common letter in the English alphabet
  - The least frequently used letters are ‘j’, ‘x’, ‘q’ and ‘z’
- Frequency analysis can always be relied on to crack the cipher
- In World War II the German’s invented the Enigma coding machine to transmit information
- This motivated the British to create one of the first cryptanalysis machines so that they could crack the code

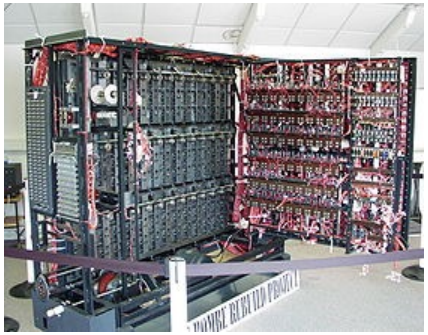
Letter	Frequency
A	8.17%
B	1.49%
C	2.78%
D	4.25%
E	12.70%
F	2.23%
G	2.02%
H	6.09%
I	6.97%
J	0.15%
K	0.77%
L	4.03%
M	2.41%
N	6.75%
O	7.51%
P	1.93%
Q	0.10%
R	5.99%
S	6.33%
T	9.06%
U	2.76%
V	0.98%
W	2.36%
X	0.15%
Y	1.97%
Z	0.07%

Breaking the cipher

- Just try all the possible combinations and look for this shape
- Apply mathematics and statistics to limit the search space



Bombe



- Alan Turing designed the Bombe computer in Bletchley Park which successfully deciphered German military communications (involving 1600 valves)
- Bombes used several stacks of rotors spinning together to test multiple hypotheses about possible setups of the Enigma machine, such as the order of the rotors in the stack
- The Germans generally changed settings each day at midnight; the British goal was to find the new settings before the day was out, preferably by noon
- With a motor spinning at 120 RPM, all permutations could be tested in under 6 hours. On average, it took half that time to find the correct match

World War II

- By 1945 all Nazi Enigma traffic could be decrypted within a day
- The British used this to turn the tide of the German U-boat attacks on ship convoys which were having a devastating effect
- Allied ships could now avoid U-boats because they knew exactly where they were
- 700 U-boats were sunk and 30,000 sailors died
- The Germans never suspected anyone would bother to crack the codes

Type VII U-boot

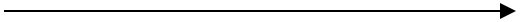
Cryptography ideals

- Ideally, a message should be secure even if the enemy knows everything about the system
- There should be no practical way to crack the system without knowing the key
  - This is known as Kerckhoff's principle
- Even if the system is decipherable in theory, it should remain impossible in practice
- Also, it should be convenient for two communicating parties to set up a shared key

Simple Example

- Bob and Alice want to communicate but don't want anybody else to be able to read the message
- Alice tells Bob her secret key that she uses to encode the message
  - Secret key is → advance each letter one place in the alphabet
- Now when Alice sends Bob a message, she can be sure that it is secure because the secret key is needed to decode it and nobody else will have it

Message



Ifmmp cpc!

Example

- But then Alice's friend Claire wants to send her a message
- Alice tells her the secret key so she can encrypt the message
- Claire encrypts the message using Alice's secret key and sends it to her

Message

Cpc’t b hpctijuf!

Bob is listening in!

Problem

- Bob is listening in and can obtain the encrypted message (ciphertext)
- He also knows Alice's secret key since she has made it available to him
- The major problem here is that everybody Alice communicates with knows her secret key and therefore all these people can decrypt her messages
- So much for security – Alice will lose all her friends!

Key distribution

- One solution would be to agree on a separate key for every single communication
- But how do you arrange a secret key without sending unencrypted messages?
- For most of the history of cryptography, keys had to be arranged beforehand using a secure, non-cryptographic method
  - Face-to-face meeting
  - Trusted courier
  - Shave a slave's head, tattoo it, let the hair grow back, and send the slave over to the other person (the Romans did this)
- Having to prearrange secret keys has significant practical difficulties

Better Idea

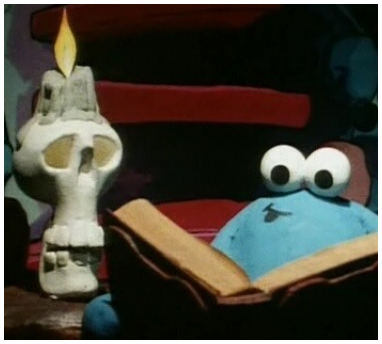
- Public key cryptography solves these problems
- The system allows users to communicate securely over an insecure channel without having to agree upon a shared key beforehand
- The main idea is that the encrypting key and decrypting key are separate
- People know how to encrypt the messages to send to Alice but nobody except Alice can decode them since she is the only one with the decrypting key

A one-way system

- This sounds like magic

- How can you possibly have a system where you can encode a message easily but not decode that same message?
- Surely to decode it you just do the opposite of the thing you did to encode it (e.g., subtract one letter of the alphabet rather than add one)?
- We need a one-way system – an irreversible process

Trapdoor

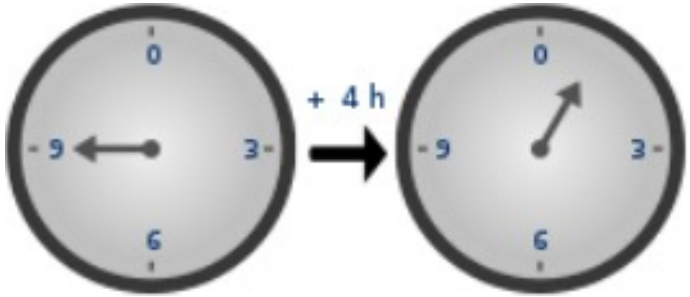


An asymmetric cipher depends on a trapdoor

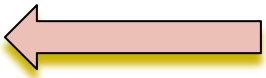
- Easy to do but very hard to undo
- Trapdoor processes come from number theory
- Two popular asymmetric ciphers are RSA and ElGamal
- RSA trapdoor: compute the product of two large primes
- Numbers are easy to multiply but hard to factor
- ElGamal trapdoor: modulo arithmetic

Modulo Arithmetic

- What time is six hours after ten o'clock?
  - Answer: Four o'clock!
- What are you actually doing here?
  - There are only 12 hours on the clock
  - If you go past 12 you go back to 0
  - (6+10) % 12 = 4



- The modulo operator means that every time you reach the modulus you go back to 0 – just like the wrap-around array
- Essentially you divide the number by the modulus and the remainder is your answer
  - 61 % 3 = 1
  - 9 % 8 = 1
  - 49 % 25 = 24

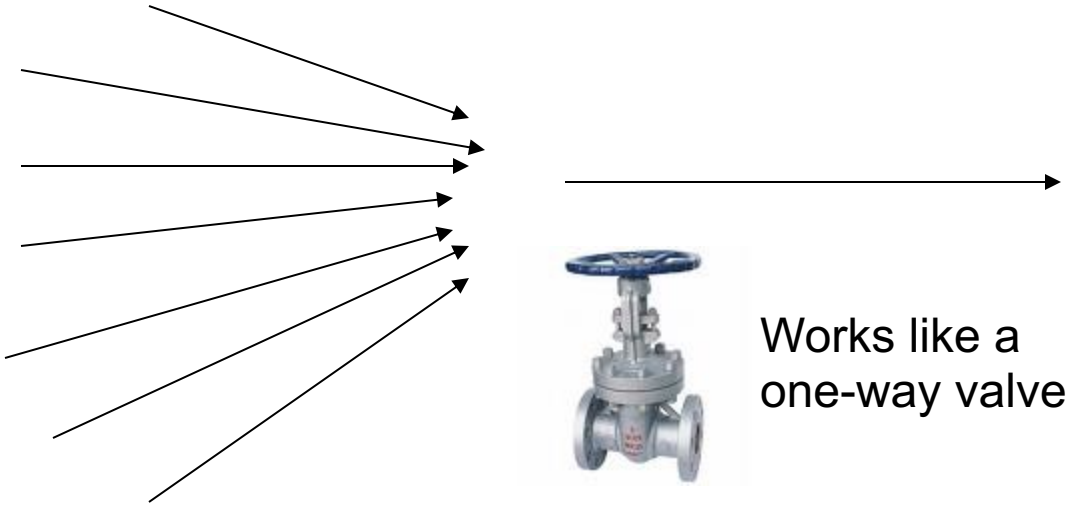


Going in Reverse

- What is six hours before three o' clock?
  - Answer: 9 o' clock
- This is easy because times can only vary between 0 - 12
- However, in many cases we can't be sure how high the value was before it was moduloed
- For example, (x + 1) % 5 = 2, solve for x
  - x could be 1
  - Also 6, 11, 16, 21, 26 etc.

One way

- We can use this to create a system where many different input values can give the same output value
- We can know how the output was generated but still not be able to figure out what the input was to generate that output
- This can be used to create an asymmetric cipher!



Public Key Cryptography

- Public key cryptography, also known as asymmetric cryptography is a form of cryptography where a user has a pair of cryptographic keys
- The private key is kept secret while the public key is widely distributed
- The keys are related mathematically but the private key cannot be derived from the public key
- A message encrypted by the public key must be decrypted by the corresponding private key

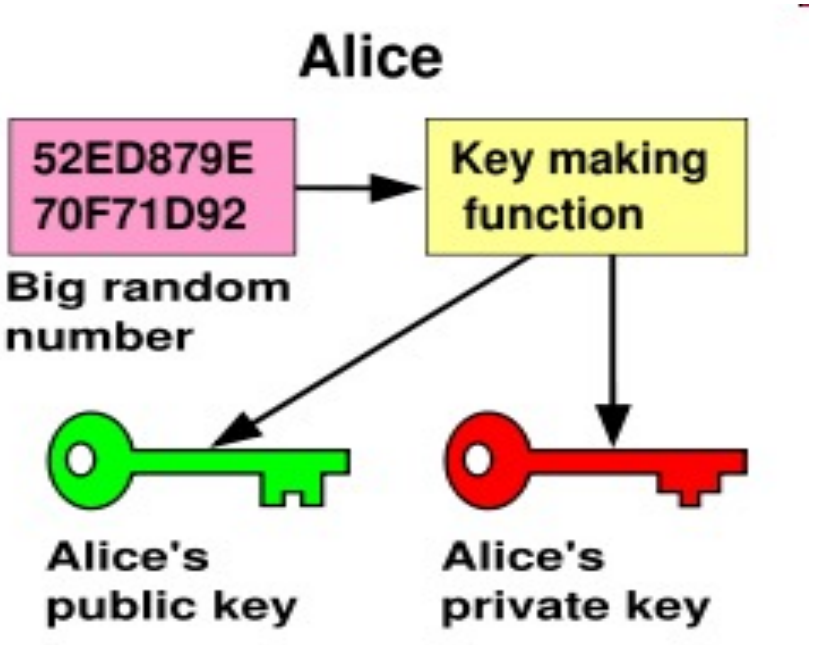
Analogy

- Imagine a locked letterbox with a slot
- The letterbox is exposed and accessible to the public
  - Its location is public, and anyone can put a letter into the slot
  - The letterbox location is the public key

- However, only the person who possesses the appropriate key can open the letterbox and read the messages
  - This is the private key

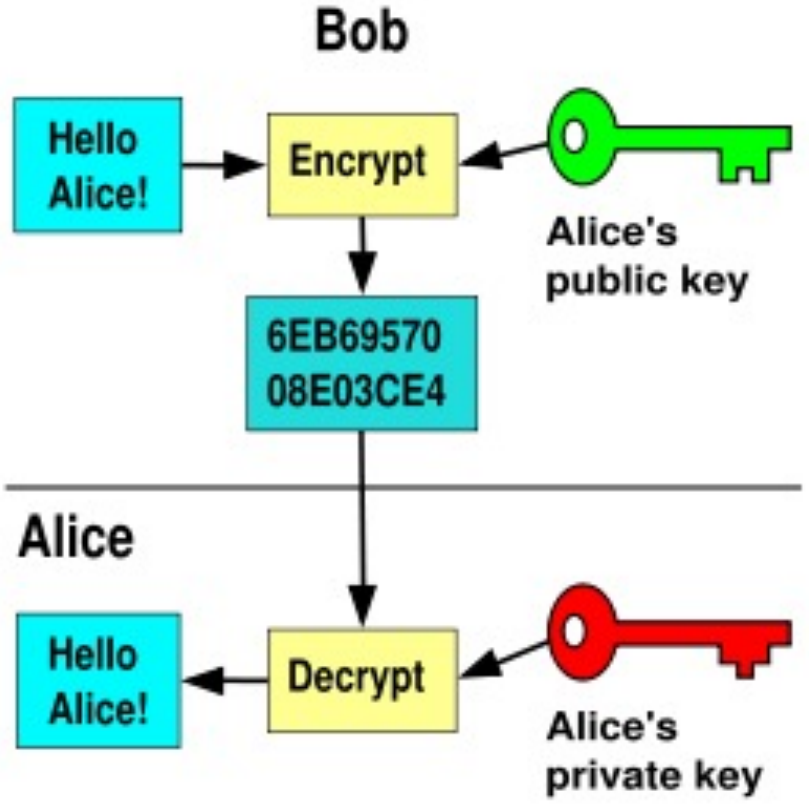
Key Generation

- Alice uses a big random number to generate a related public and private key
- She keeps the private key to herself and publishes the public key so people can send her messages
- The level of security depends on the secrecy of the private key and how protected it is



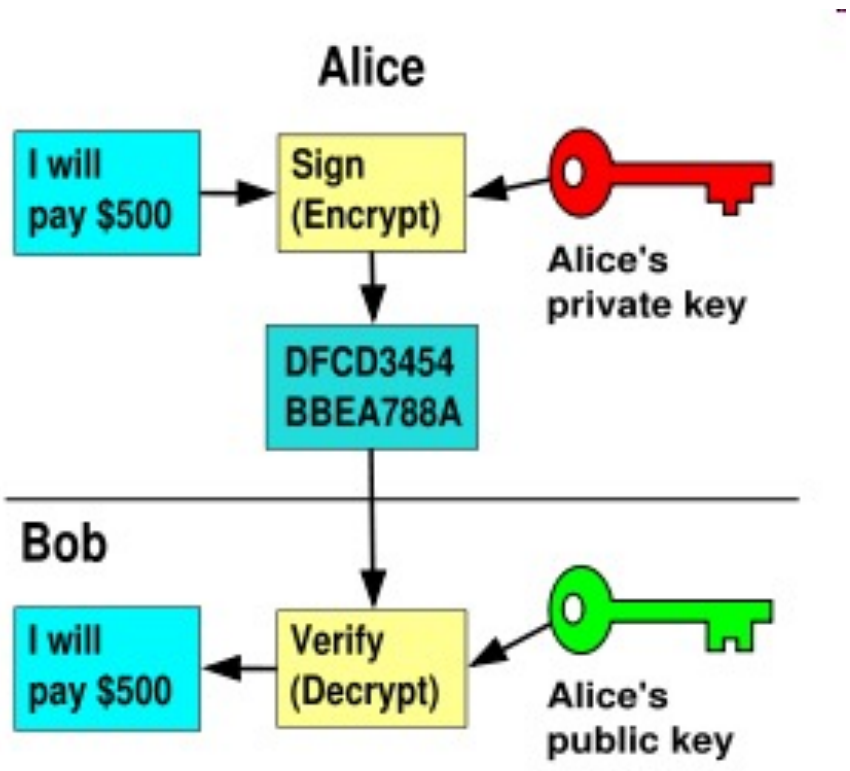
Encryption

- Bob uses Alice's public key to encrypt a message
- He adds an extra layer of encryption on top of Alice's public key
- Nobody can reverse Bob's additional encryption
- However, Alice's private key has been chosen so that is it able to reverse Bob's additional encryption without needing to know what that original encryption was



Signing a message

- The same system can be used backward for signature verification
- Using a private key to encrypt (thus signing) a message anyone can check the signature using the public key



Problems

- Hackers might try to pick Bob or Alice's lock
- All public-key schemes are susceptible to brute force attack
- Attacks are impractical if the amount needed to succeed is beyond the means of the hacker
- Work factor can simply be increased by choosing a longer key that makes an attack impractical
- Mathematical discoveries and clever heuristics can improve the brute force attack but only marginally
  - $O(n) \rightarrow O(\sqrt{n})$
- Badly chosen keys (e.g., non-prime numbers) can also make it much easier to crack a private key

ElGamal

- The ElGamal algorithm is an asymmetric key encryption algorithm for public key cryptography used with a wide range of cryptosystems
- It was described by Taher Elgamal in 1984, an Egyptian cryptographer
- There are three components
  - The key generator
  - The encryption algorithm
  - The decryption algorithm

Discrete Log Problem

- The ElGamal system is based on the discrete log problem
  - $7^x \bmod 150001 = 66436$, what is x?
- If there was no modulus, then it would be simple log problem
  - $x = \log_7 66436$
- However, the modulus creates a trapdoor
- There is no clear strategy to figuring out what x is other than trying all the possibilities
- Log problems with a modulus are called discrete log problems
- It is easy to verify an answer is correct but difficult to find the answer – this creates a trapdoor

Key Generator

- Alice picks a generator number g
- She also picks a modulus p
- Now Alice picks a random private key x that lies somewhere between 0 and p
- Using these three numbers Alice computes g^x modulo p
- Alice then publishes (p, g, g^x mod p) as her public key
- Others know that the generator has been raised to some power x and then moduloed to give the result
- However, they are unable to figure out what the power is

Example

- Alice picks the generator g as 7
- She picks the prime modulus p as 150001
- She then chooses a secret key x as 113
- Now she calculates g^x mod p which is 66436
- Alice's public key is therefore (150001, 7, 66436)
- If anybody wanted to obtain her secret key, they would have to figure out

$$7^x \bmod 150001 = 66436, \text{ what is } x?$$

- To get x you need to solve the discrete log problem

Encryption Algorithm

- Bob has Alice's public key (p, g, $g^x \bmod p$)
- Bob converts his message m into a number between 0 and p
- He then picks a random number y again between 0 and p
- Bob now calculates $g^y \bmod p$ and $m.(g^x)^y \bmod p$
- Bob sends the cipher ($g^y \bmod p$, $m.g^{xy} \bmod p$) to Alice
- Bob can't figure out Alice's private key, but he has encrypted his message by raising Alice's public key to the power of a random number
- Only Alice's secret key can reverse this process.

Example

- Bob chooses a password as 131513
- He wants to communicate this shared password to Alice without eavesdroppers being able to obtain it
- Bob now picks a random number, say 1000
- He has Alice's public key (150001, 7, 66436)
- He, therefore, calculates 7^{1000} mod 150001 which is 90,429
- He also calculates 131513. 66436^{1000} mod 150001 which is 57,422
- Now he sends these two numbers (90429, 57422) to Alice

Decryption Algorithm

密码

- Alice receives the cipher from Bob,

  (g^y mod p, m.g^{xy} mod p). we'll call these numbers c_1 and c_2.
- She uses her secret key to extract the message m.
- This is easy – she just computes c_2 / c_1^x
- And how does this work?

$$\frac{c_2}{c_1^x} = \frac{m \cdot g^{xy}}{g^{xy}} = m.$$

Example

- Alice receives the cipher from Bob, (90429, 57422)
- She uses her secret key to compute $1 / c_1^x$
- It's best to avoid division when using modulo arithmetic
- $1 / c_1^x$ is the same as c_1^{p-1-x} which is 90429^{149887} mod 150001 which is 80802
- Now she multiplies 80802 by c_2 mod 150001 to yield the original message 131513
- This shared piece of information can now be used to encrypt documents sent between them

Analogy

- Alice has told everyone what her lock is but only she has the key
- Bob attaches Alice's lock to his message but he also adds his own lock on top of Alice's lock
- Alice gets the message, uses her key on the original lock and both locks come off
- The double lock is necessary or else the message could be decrypted by an eavesdropper
- Bob must add an extra level of encryption that only he knows
- The genius of the system is that Alice is able to decrypt the cipher without needing to know what Bob has done

Eavesdroppers

- Anyone listening in will obtain the message that is sent by Bob ($g^y \bmod p$, $m.g^{xy} \bmod p$)
- They will also know Alice's public key $g^x \bmod p$
- This is not enough to obtain m
- In order to obtain the message m, you need to know either
  - g^x and y
  - x and g^y
- The only people who have these are Bob and Alice
- They only ever transmit one of the pair
  - Alice transmits g^x and Bob transmits g^y

Cracking the Private Key

- Everything we have looked at so far concerns just sending a message
- What if we want to crack Alice's private key to listen in on her messages?
- We have to solve the discrete log problem
  - $7^x \bmod 150001 = 66436$, what is x?

Brute Force Search

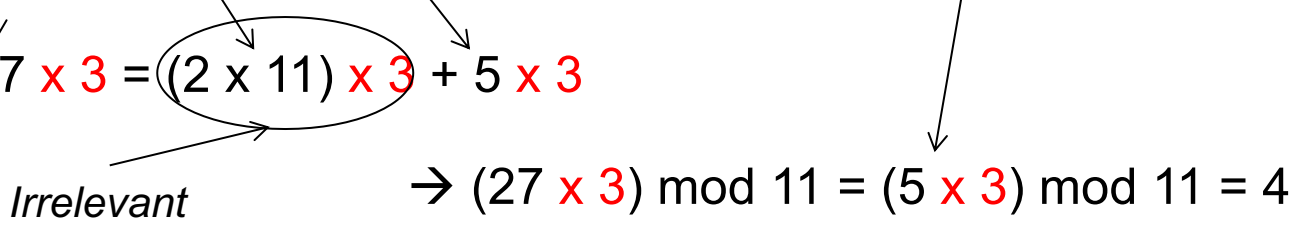
- The most intuitive way to crack this is to simply try every value for x between 0 and 150001
- Keep raising the value of x until we find one that yields 66436
- If we try the values up to 113, we will find that this is the secret key, and we will be able to decrypt Alice's messages
- Keys can be cracked in O(√n) time using the Baby Step Giant Steps algorithm, provided an O(1) searching algorithm is used
- The disadvantage is that the memory requirement is O(√n)
- This is an example of a runtime/memory trade

Big Numbers

- These kinds of numbers will exceed the capacity of a typical calculator
- Imagine we're trying to find 3^{100} mod 11
- We could calculate 3, 9, 27, 81, 243, 729 … but the numbers will get ridiculously big very quickly
- To get around the problem we can apply the modulus during the calculation with no effect on the answer
  - every time the number goes beyond 11, just modulo it and take it from there
  - 3, 9, 5 (27 mod 11) , 4 (15 mod 11) , 1 (12 mod 11) , 3…

Big Numbers

- Why is this allowed?
  - 27 = (2 x 11) + 5
  - 27 x 3 = (2 x 11) x 3 + 5 x 3



  - Only the remainder has any significance because when the multiples of 11 are multiplied they are still multiples of 11 and don't contribute
  - We only need to multiply the modulus remainder each time

Example

- Calculate $7^{113} \bmod 150{,}001$
- The trick is to break down 113 into simpler units
  - $7 \bmod 150001 = 7$
  - $7^2 \bmod 150001 = (7 \bmod 150001)^2 = 7^2 = 49$
  - $7^4 \bmod 150001 = (7 \bmod 150001)^2 = 7^2 = 2{,}401$
  - $7^8 \bmod 150001 = 64{,}763$
  - $7^{16} \bmod 150001 = 68{,}208$
  - $7^{32} \bmod 150001 = 50{,}249$
  - $7^{64} \bmod 150001 = 145{,}169$

- $113 = (1 \times 64) + (1 \times 32) + (1 \times 16) + (1 \times 1)$

Example

- 113 = (1 x 64) + (1 x 32) + (1 x 16) + (1 x 1)
- 7^{113} mod 150,001 = (145,169 x 50,249 x 68,208 x 7) mod 150,001
- 7^{113} mod 150,001 = 66,436
- Of course, you can break 7^{113} up into any smaller powers that you like
- For example $7^{113} = 7^{50} \times 7^{50} \times 7^{13}$
- Find out the largest powers that your calculator can handle and use those

How do I do this on my calculator?

- Given $7^4 \bmod 150001 = 2{,}401$ find $7^8 \bmod 150001$
  - $7^8 \bmod 150001 = 7^4 \times 7^4 = 2{,}401 \times 2{,}401$
  - $2{,}401 \times 2{,}401 = 5{,}764{,}801$
  - $5{,}764{,}801 / 150{,}001 = 38.43175046.....$
  - $38.43175046 - 38 = 0.43175046$ .....
  - $0.43175046 \times 150{,}001 = 64{,}763$
  - $7^8 \bmod 150001 = 64{,}763$

Implementation

- How do we write a method for calculating modulus powers?

```
public static int modPow(int number, int power, int modulus){
     int result =1;
     for(int i=0;i<power;i++){
            result=result*number;
            result=result%modulus;
     }
     return result;
}
```

- This loop approach is extremely inefficient because it is O(n)
- We have to calculate every preceding modPow before arriving at the one we want
- This essentially defeats the purpose of the trapdoor because computing with the key is supposed to be far quicker than trying to hack the key

Implementation

- We can write a recursive O(logn) method:

```
public static int modpow(int number, int power, int modulus){
        if(power==1)
            return number%modulus;
        else if (power % 2 ==0) {
            long halfpower=modpow(number, power/2, modulus);
            return (halfpower*halfpower) % modulus;
        }else{
            long halfpower=modpow(number, power/2, modulus);
            return (halfpower*halfpower*number) % modulus;
        }
}
```

- This is called exponentiation by squaring

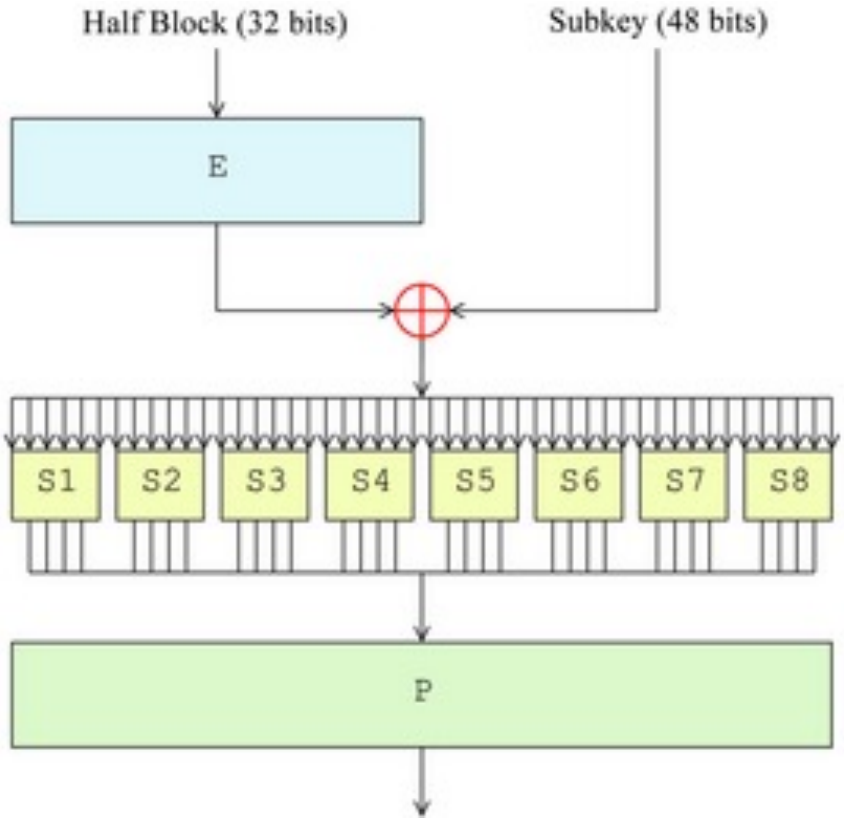
$$\text{Power}(x, n) = \begin{cases} 1, & \text{if } n = 0 \\ x \times \text{Power}(x, n-1), & \text{if } n \text{ is odd} \\ \text{Power}(x, n/2)^2, & \text{if } n \text{ is even} \end{cases}$$

Data Encryption Standard

- Public key cryptography is only able to send a single piece of information which is a number between 0 and the modulus
- Therefore, it can't be used to send a large text file
- Instead, public-key cryptography is usually used to set up a shared key to be used with a more efficient symmetric key
- Nobody knows the shared symmetric key except Alice and Bob
- Information is then encrypted and decrypted using the symmetric key which can be used on large chunks of text
- The most commonly used is the Data Encryption Standard (DES)

Data Encryption Standard

- DES was selected as the official encryption standard for the USA in 1976
- It uses a 56-bit key size which may now be too small as keys can be cracked within 24 hours
- DES is a block cipher – it takes a fixed-length string of bits and transforms it into a ciphertext bitstring of the same bit length



Data Encryption Standard

- The algorithm has 16 identical stages of processing where bit sequences are translated
- Each 32-bit half-block is expanded to 48 bits by adding extra random bits and then transformed by 16 48-bit subkeys derived from the main key.
- After this mixing process, 6-bit pieces are put into substitution boxes which produce 4-bit outputs
- Finally, the output of the 32 substitution boxes is re-arranged according to a fixed permutation

Big Numbers

- The bigger the modulus is, the more secure your system will be, since the private key will have a greater possible range
  - The recommended length is 1024 bits
  - This is a number with over 30 digits!
- Big numbers can be implemented in Java as long
  - The maximum length of a long is 19 digits
  - Special power and multiplication methods must be written to avoid the numbers becoming too large
  - You must add an “l” at the end if defining a long: 285729562549l
- If you need to use bigger numbers, there is a Java class in the math package called BigInteger which takes care of all this
  - Look this class up at http://java.sun.com
  - The class also has methods for calculating mod powers

BigInteger Example

```
import java.math.BigInteger;

  BigInteger p=new BigInteger("1036853");
  BigInteger y=new BigInteger("228221");
  BigInteger g=new BigInteger("649");
  BigInteger c1 = (g.modPow(random, p));
  BigInteger c2 = m.modPow(g.modPow(random, p), p);
  System.out.println(c1.intValue());
  System.out.println(c2.intValue());
```

Cryptography

- The vast majority of asymmetric cryptosystems in operation today use either ElGamal or RSA
- These systems depend on discrete logs and factorization being one-way processes
  - The operations are easy to do but extremely difficult to undo like smashing a cup
- However, it has not been proved that discrete logs and factorization are genuine one-way processes
- It has not even been proved that one-way processes exist!
- In the future, the discovery of an efficient algorithm may cause these cryptosystems to fail, with serious implications

Questions