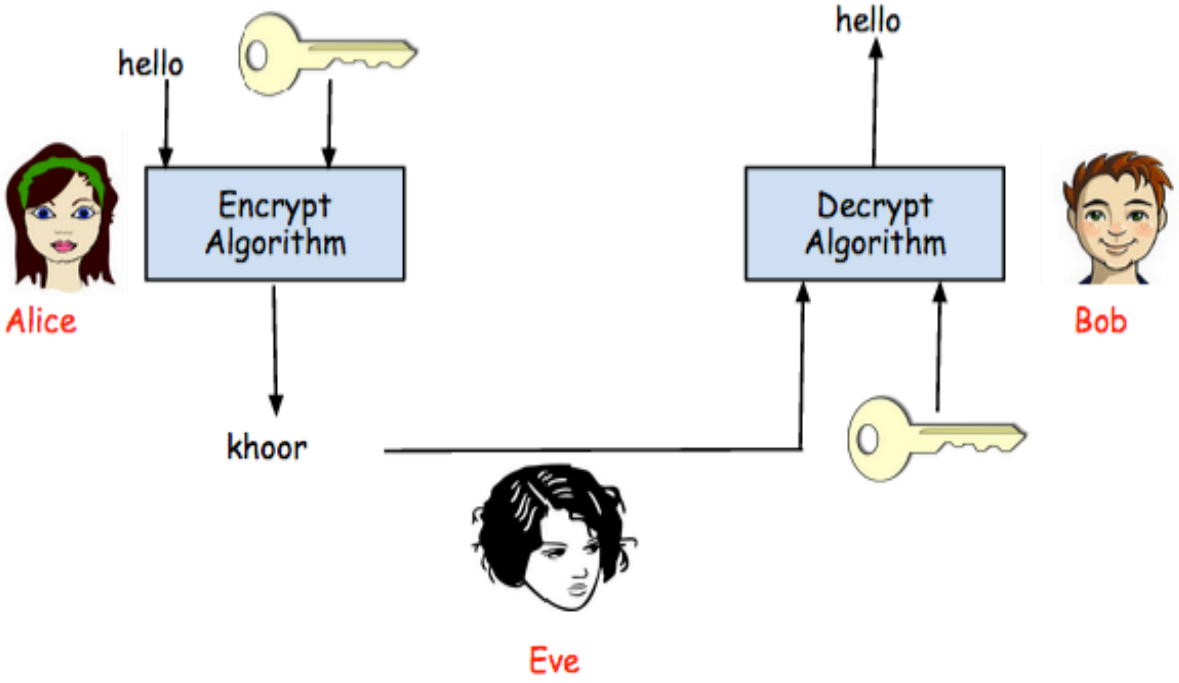
Cryptography

Basic Terminology

- *Cryptography* means *secret writing*
- *Encryption* means converting *plaintext* into *ciphertext*
  - *hello* → *khoor*
- *Decryption* means converting *ciphertext* back into *plaintext*
  - khoor → hello

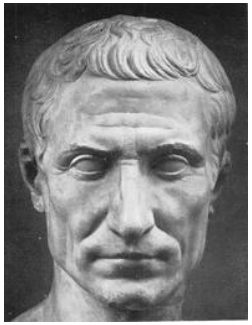
Symmetric Encryption Model

- Encryption and decryption require a *key* (can be an integer) and an *algorithm* (we'll look at four today)
- *Symmetric encryption*: the same key is used for both encryption and decryption

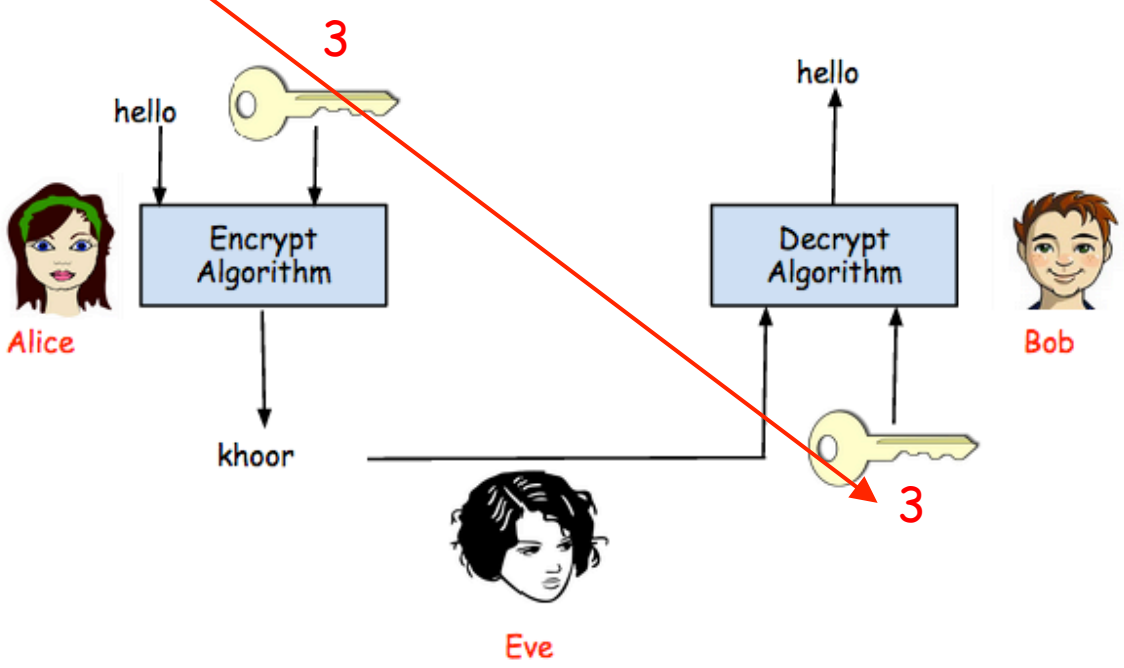


Caesar Cipher

(A substitution cipher)



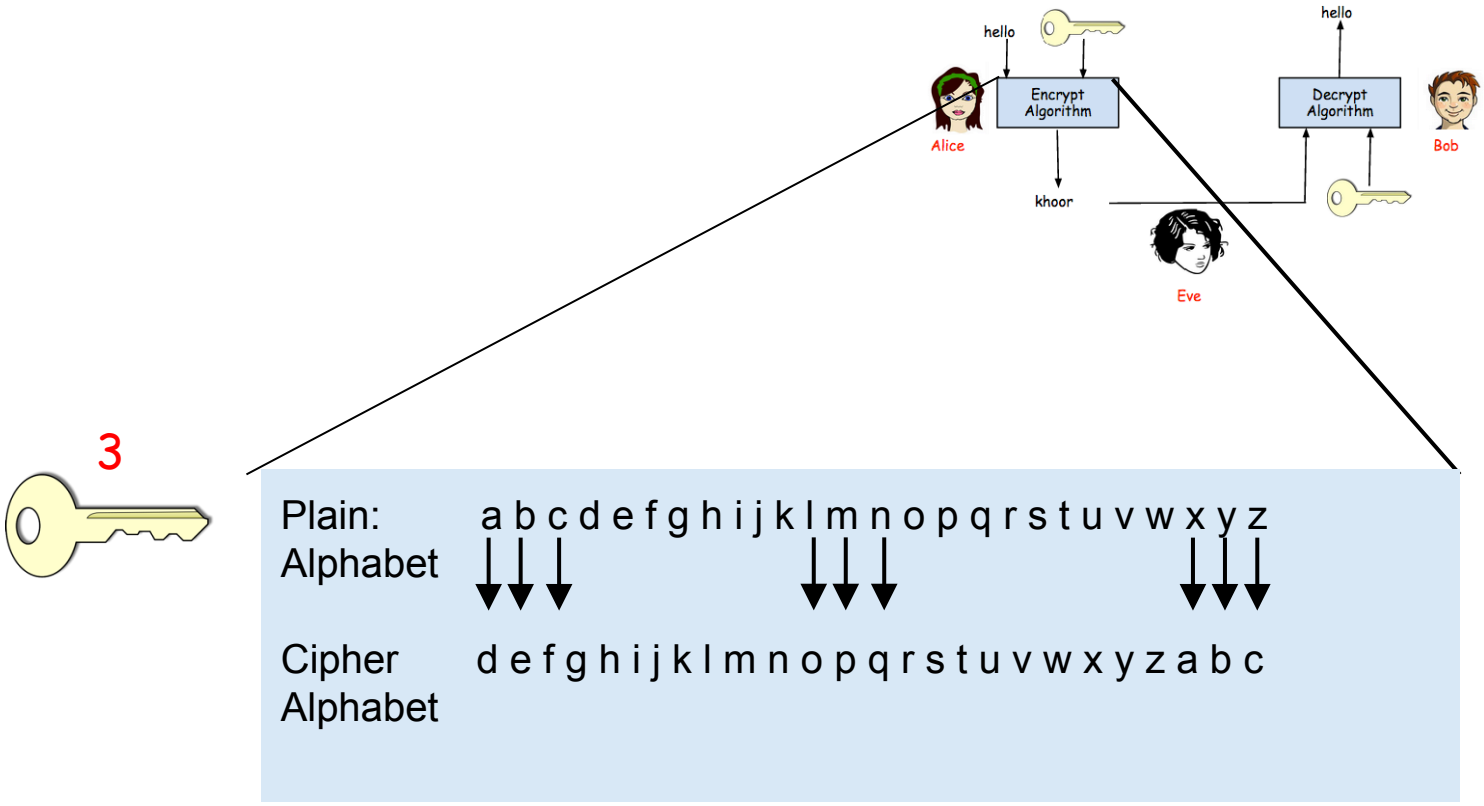
- First commonly used cipher (38 BC)
- *Key*: Shift the alphabet by *N* letters to create cipher alphabet



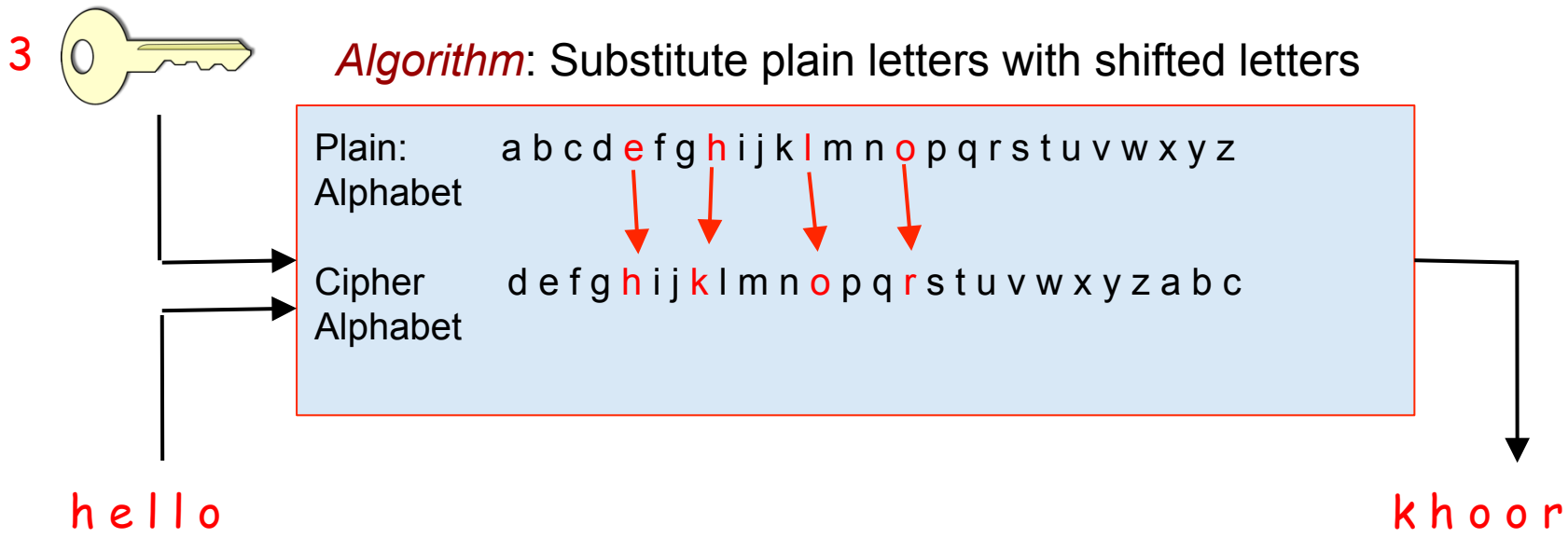
Caesar Cipher

(A substitution cipher)

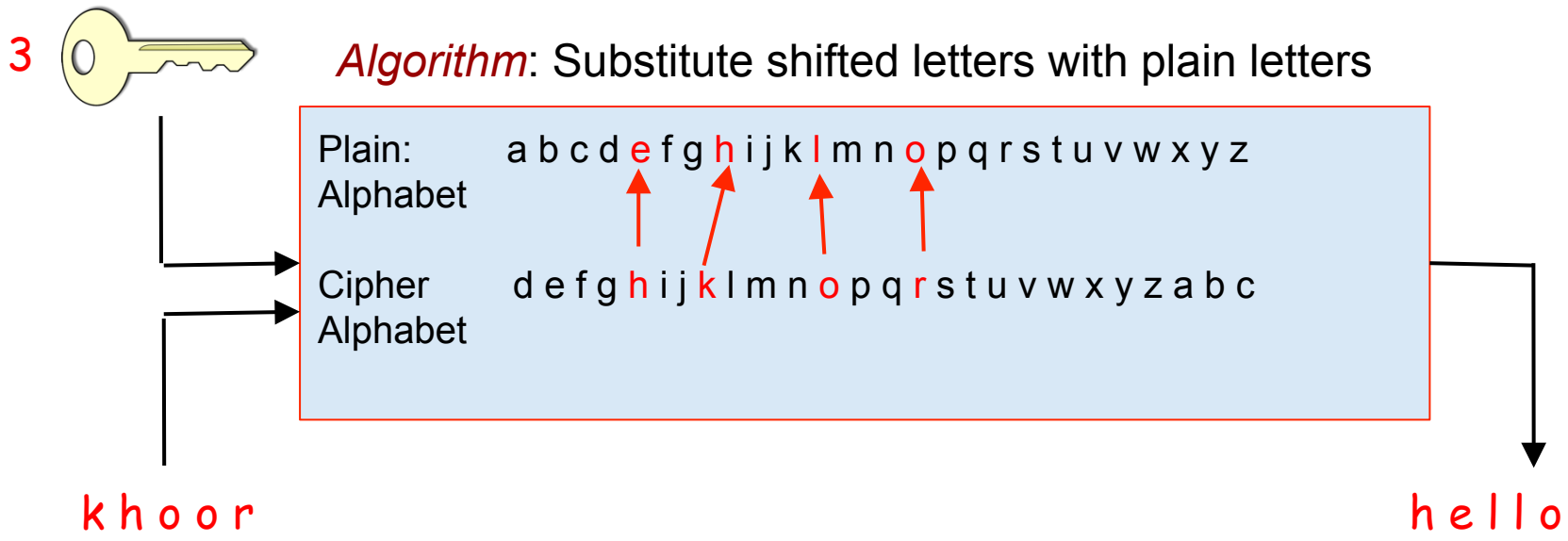
- *Key*: Shift the alphabet by *3* letters to create cipher alphabet
- *Algorithm*: Substitute plain letters with shifted letters



Caesar Encryption



Caesar Decryption

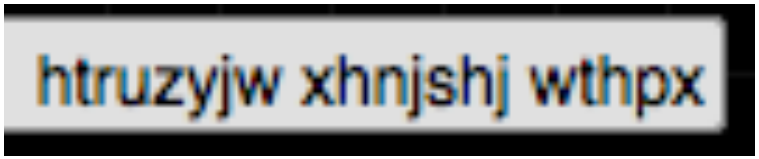


Caesar Cipher

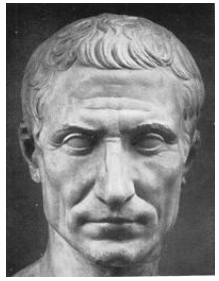
- Use the Caesar cipher to encrypt your name, use a key of 3, 1.5 minutes
- A ***brute force*** search for a Caesar key would be to test every possible key. How many keys would you have to test? Use brute force search to ***crack*** the following message. Key is 2..8, volunteers?

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
```

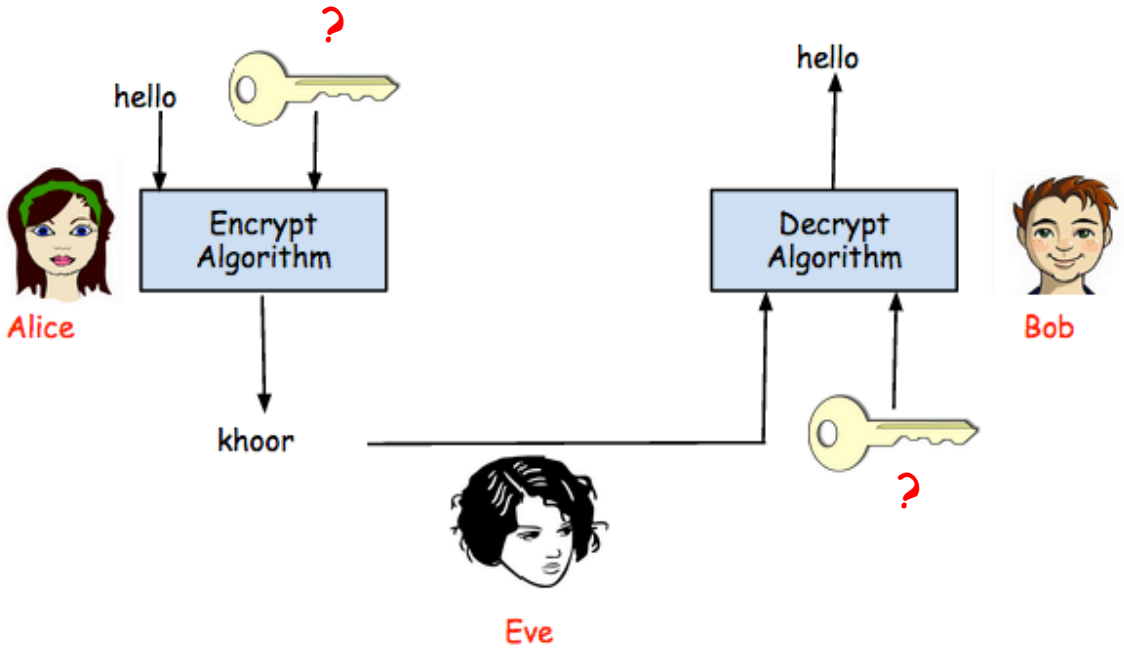
htruzyjw xhnjshj wthpx



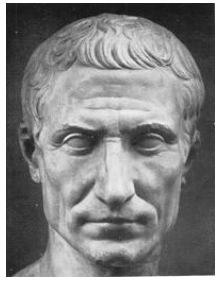
Cracking Caesar Cipher



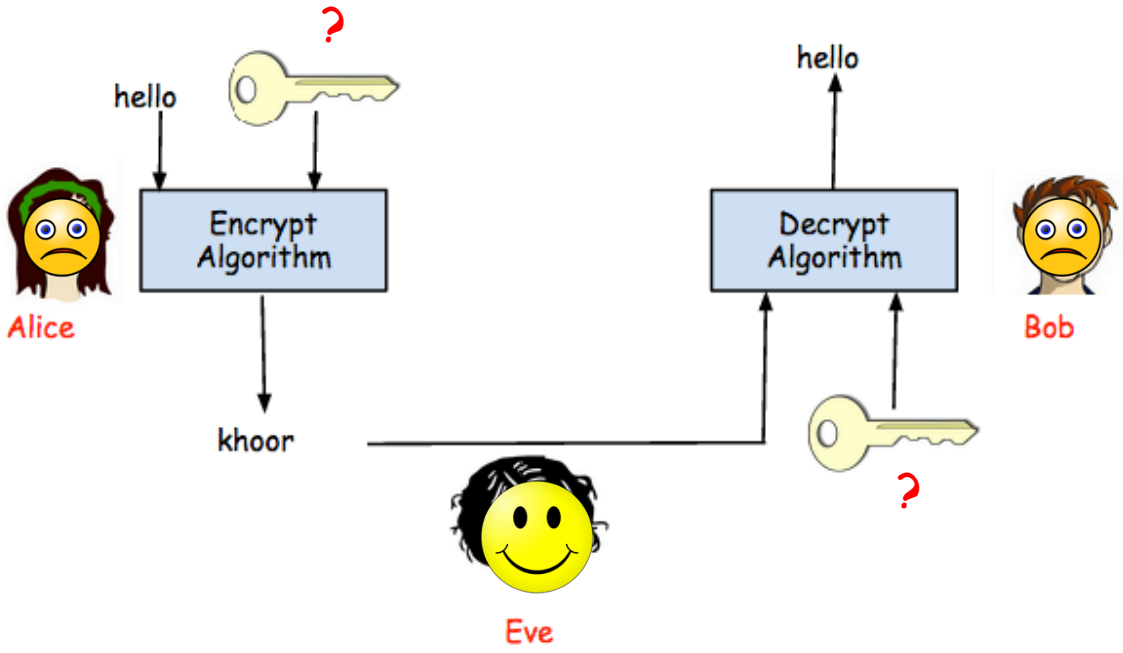
- *Brute force attack:* How many keys would Eve have to try to break Alice's message to Bob?



Cracking Caesar Cipher

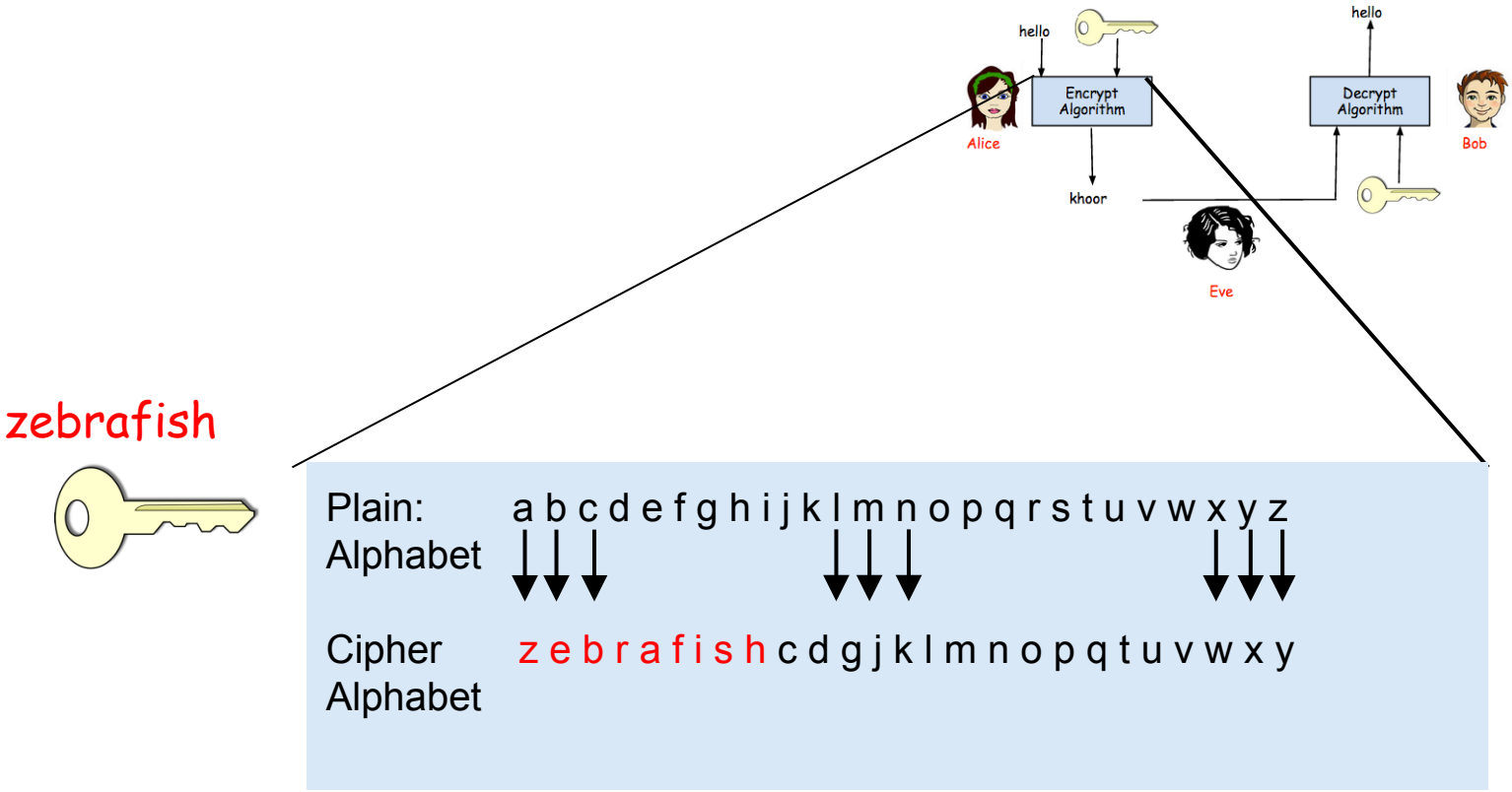


- *Brute force attack:* How many keys would Eve have to try to break Alice's message to Bob?
- *Answer*: 25 keys. Too easy!



Simple Substitution Cipher

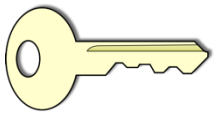
- *Key*: Randomly mix up alphabet to get cipher alphabet
- *Algorithm*: Substitute plain letters with shuffled letters



Simple Substitution Cipher

- Encrypt "good job" illr cle
- Only have to know the magic word "zebrafish"

zebrafish

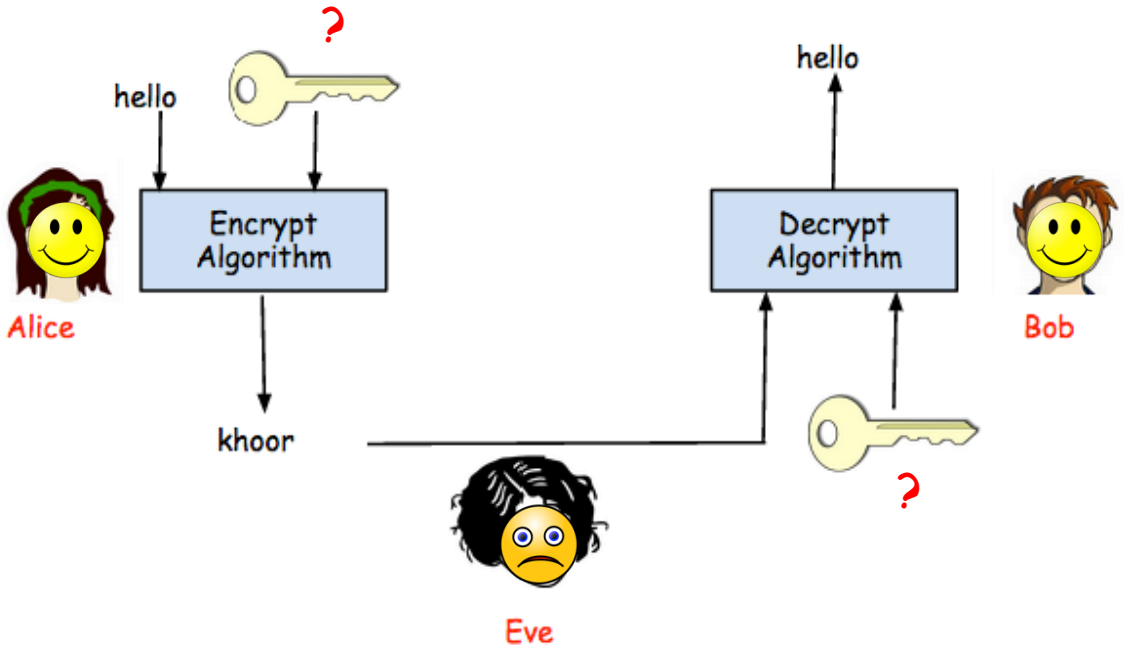


Plain: Alphabet a b c d e f g h i j k l m n o p q r s t u v w x y z

Cipher Alphabet z e b r a f i s h c d g j k l m n o p q t u v w x y

Cracking Simple Substitution

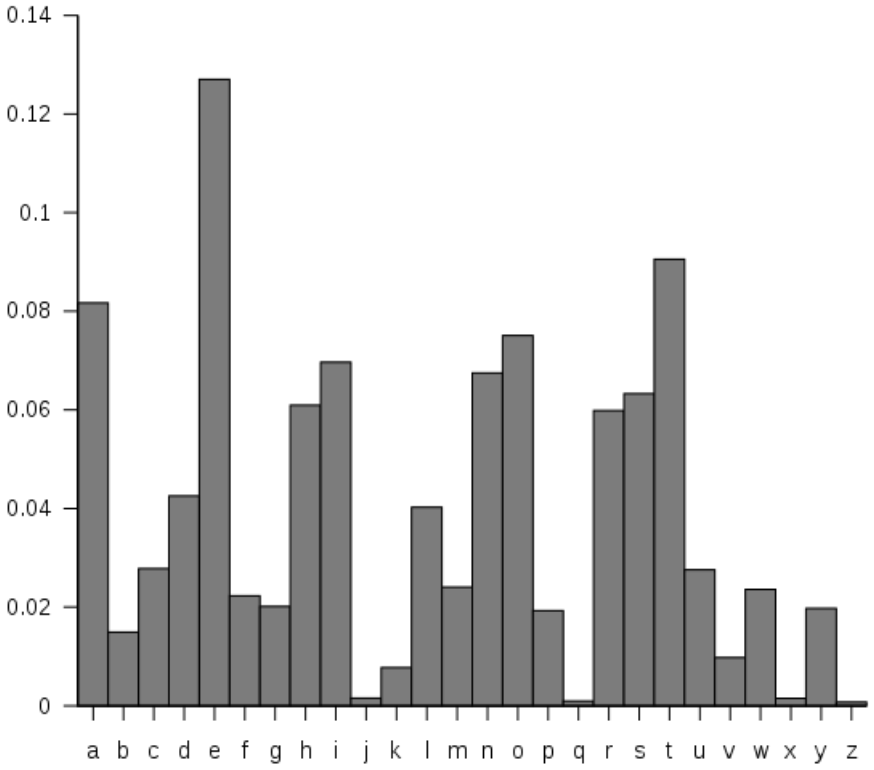
- *Brute force attack:* Eve would need 26! keys.
- That's 4.0329146e+26 keys. Too hard!



Cracking Simple Substitution

- *But, wait a minute...*

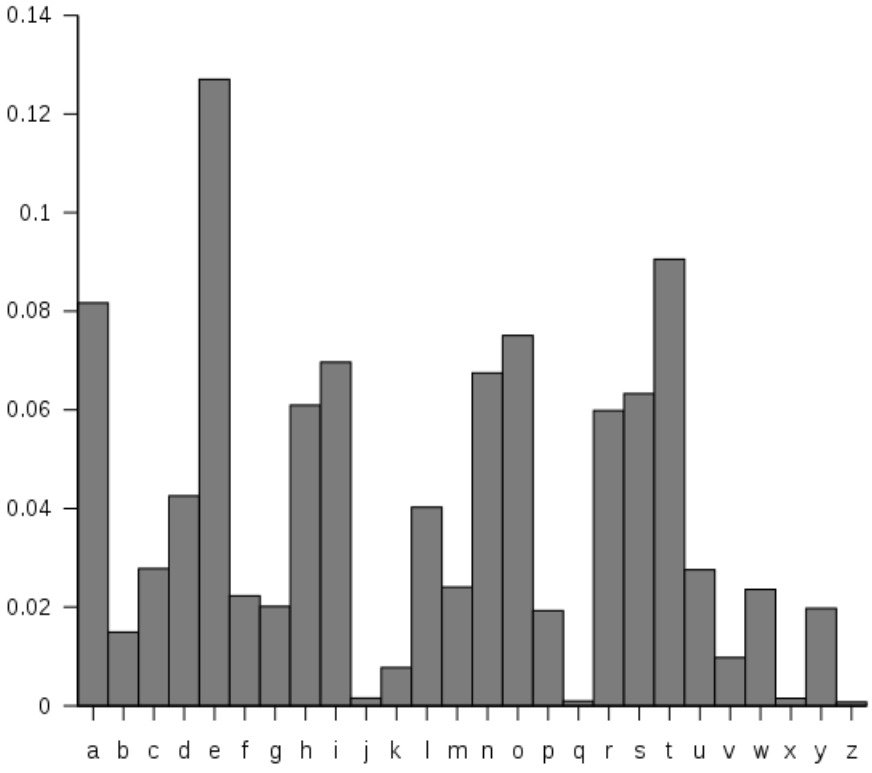
English plaintext
letter frequencies



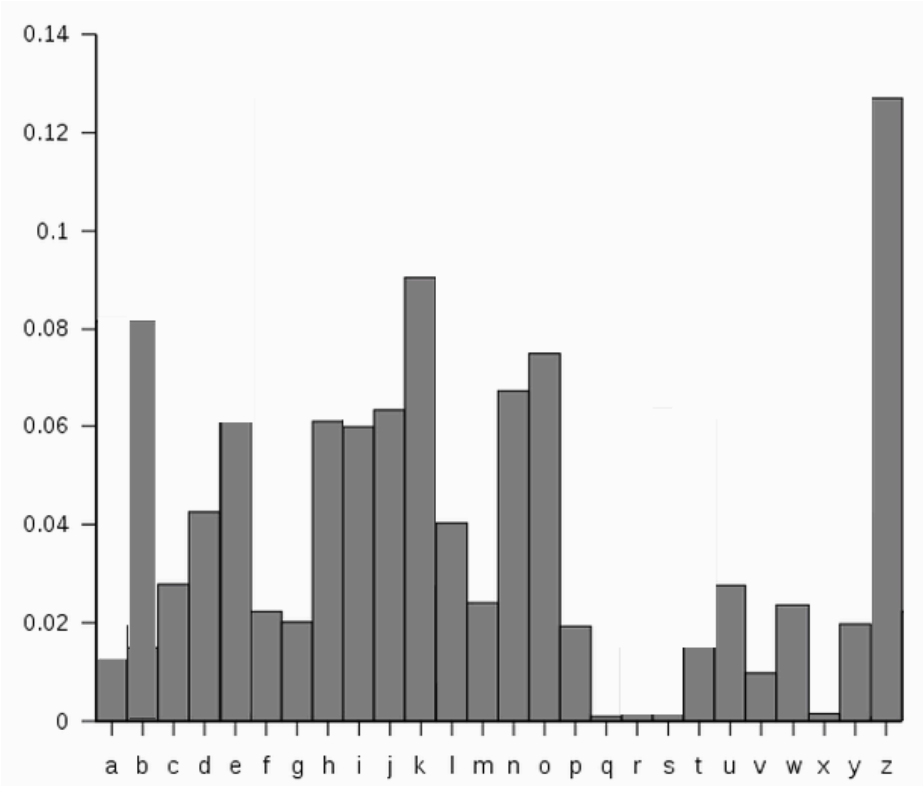
Cracking Simple Substitution

- *But, wait a minute...*

English plaintext
letter frequencies



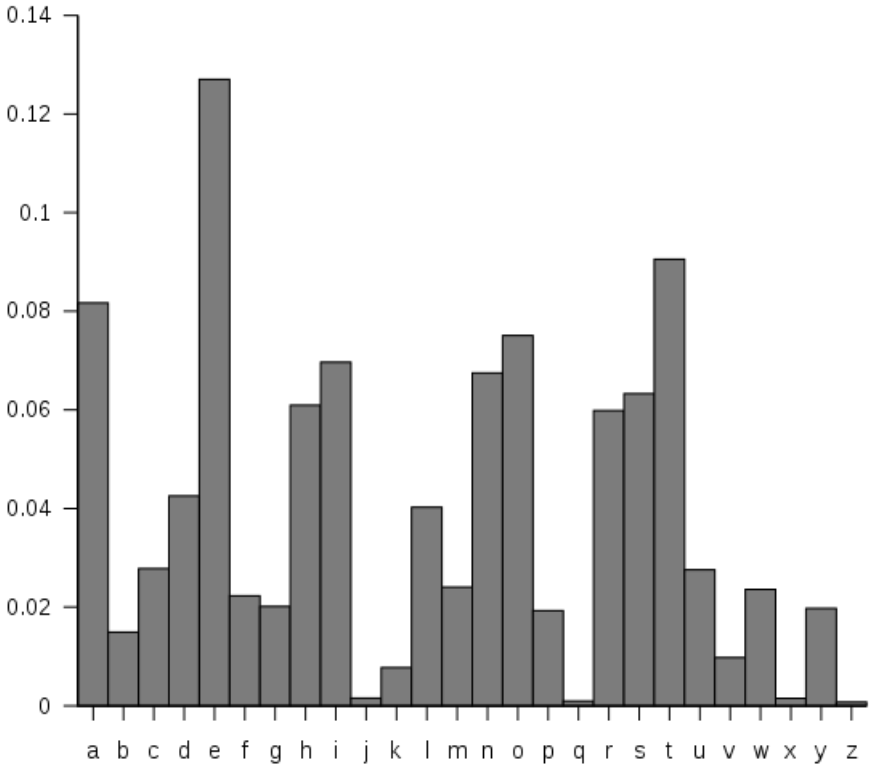
Ciphertext
letter frequencies



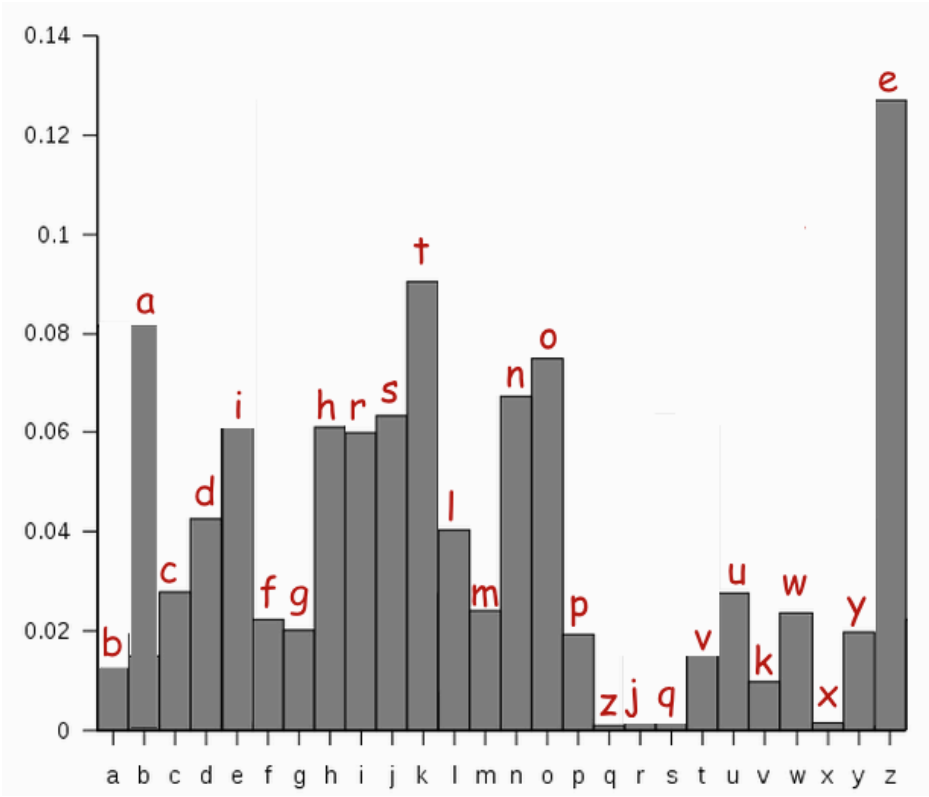
Cracking Simple Substitution

- *But, wait a minute… frequency analysis works!*

English plaintext
letter frequencies

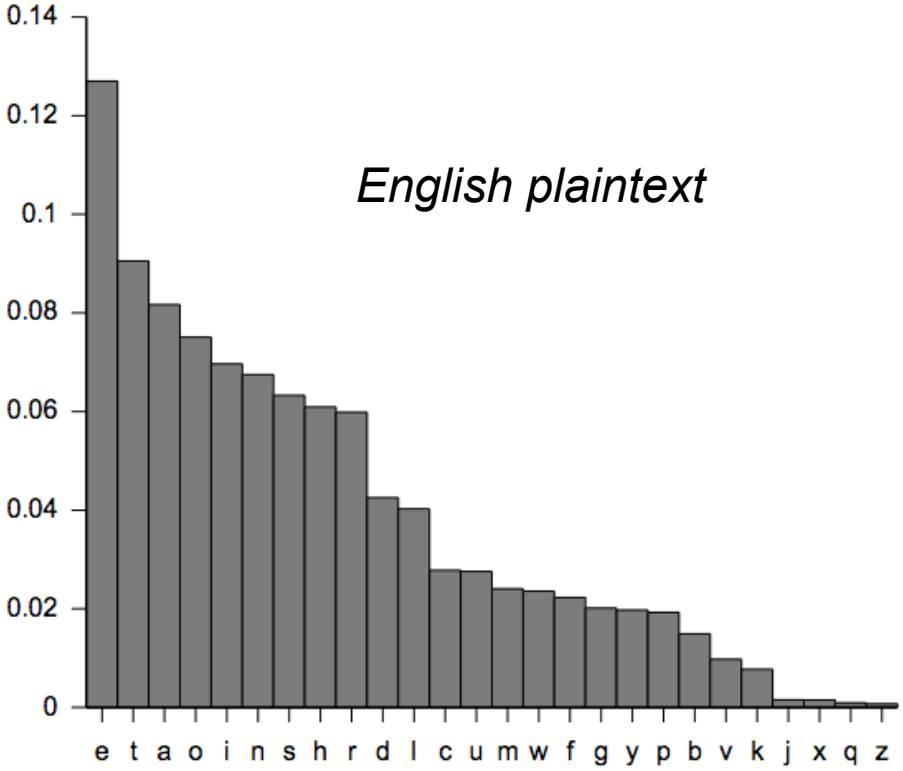


Ciphertext
letter frequencies

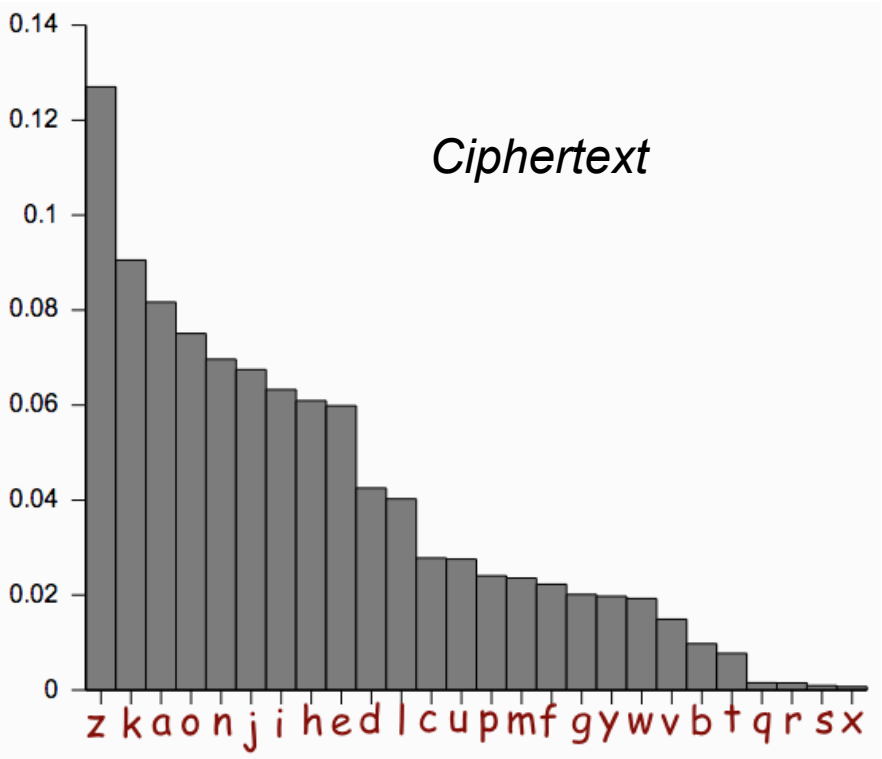


Cracking Simple Substitution

- *Can sort by frequencies*



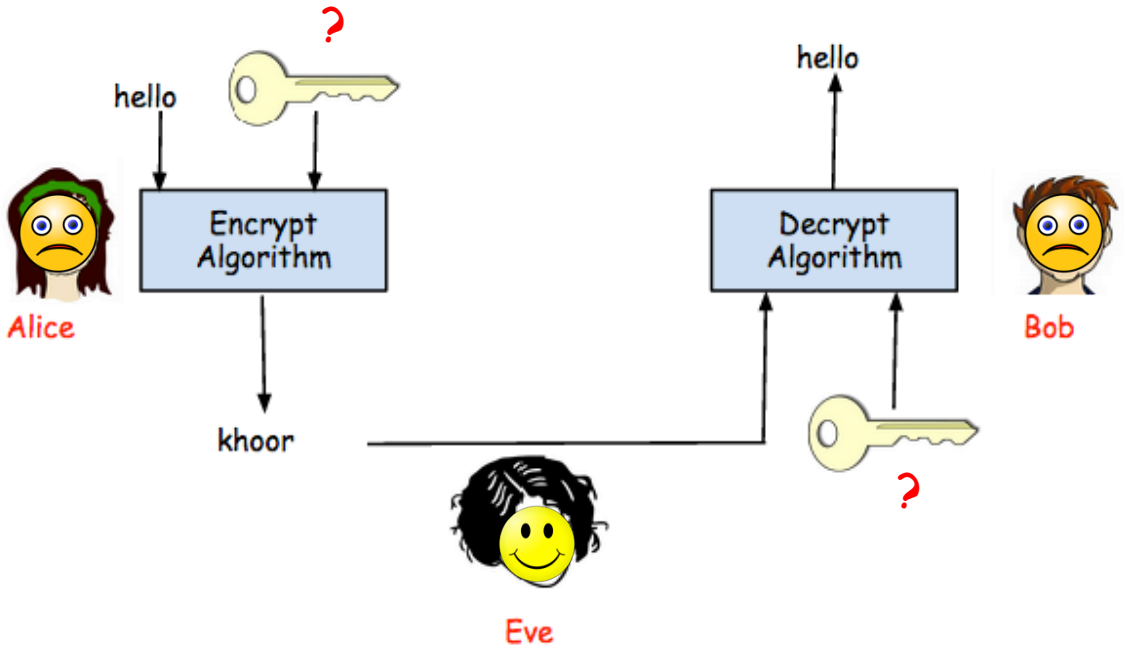
English plaintext



Ciphertext

Cracking Simple Substitution

- Eve wins … you don't need brute force
- *Frequency analysis* will break simple substitution



Frequency Analysis

- We've been talking about *substitution ciphers*, where ciphertext letters are substituted for plaintext letters
- A *transposition cipher* is one where the letters in the message are rearranged -- the alphabet is unchanged
  - Plaintext: hello world this is a transposed message
  - Transposed: olleh dlrow isiht artas sopsn semde egas
- Can you see the transposition rule?
- Transposition ciphers don't change the letter frequencies
- We can use *frequency analysis* to tell whether a text has been encrypted with a transposition or substitution cipher

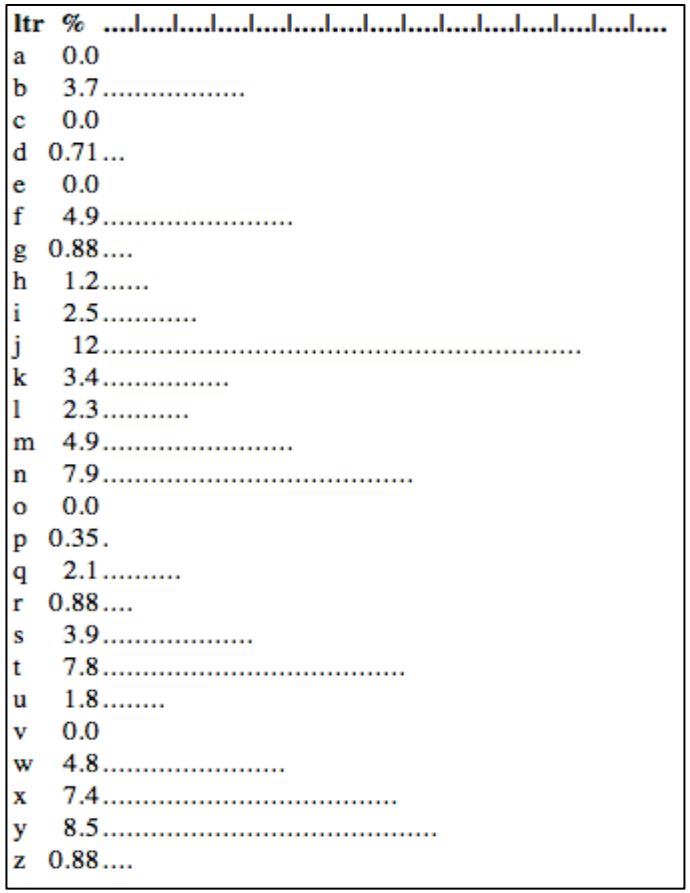
Frequency Analysis

- One of the following texts was encrypted using a *transposition cipher* and the other with a substitution (Caesar) cipher, which is which?

Text 1: nybfx ymjgj xytky nrjxn ybfxy mjbtw xytky nrjx nybfx ymjfl jtkbn xitrn ybfxy mjflj tkktt qnxms jxxn ybfxy mjjut hmtkg jqnjk nybfx ymjju thmtk nshwj izqny dnyb fxymj xjfxt stkqn lmyny bfxym jxjfx tstki fwpsj xxny bfxym jxuwn sltkm tujny bfxym jbnsy jwtki jxufn wbjm fijaj wdymn slgjk twjzx bjmfi stymn slgjk twjzx bjbj wjfqq ltnsl inwjh yytmj fajsb jbjwj fqqlt nslin wjhy ymjty mjwbf dnsxm twyym jujwn tibfx xtkfw qnpjy mjuwj xjsy ujwnt iymfy xtrjt knyxs tnxnj xyfzy mtwny njxns xnxyj itsny xgjn slwjh jnaji ktwlt titwk twjan qnsym jxzuj wqfyn ajijl wjjt khtru fwnxt stsqd

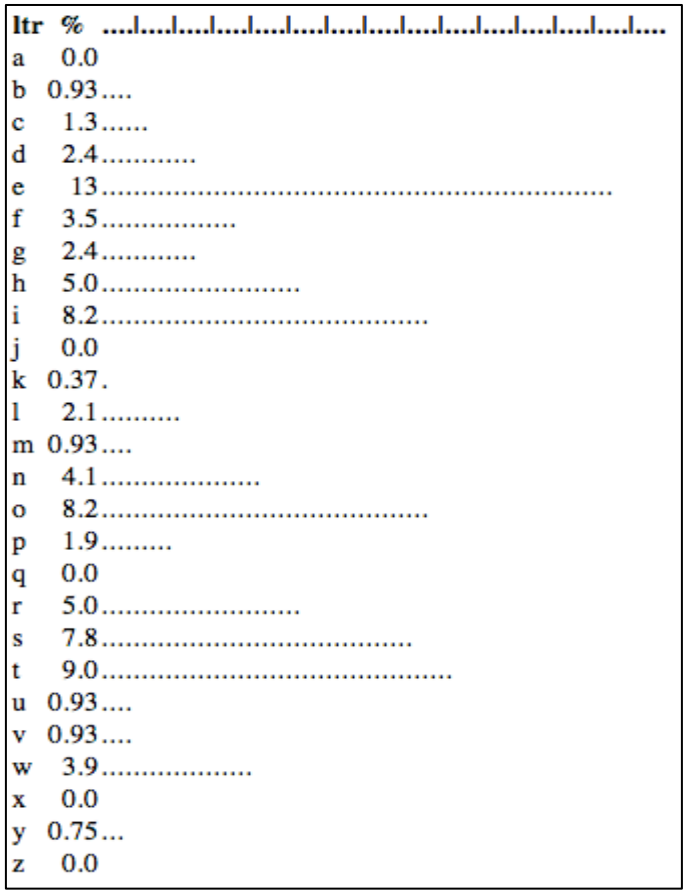
Text 2: ttbti swhot istta osmwh gflhs tsecf liaho ondia henit ahena nwtpnf ewtie fpree rhbou hnhbo uerli deovw rlode oeasr hrdsa itrei ttein ittie ntote gceoo rrits etegc psoya hsfmt sesfm iahew dtseo oiewh pheet tecir uytss sohts ssoks isero oisen oeawa vtnee watne ewagn rtenw egnit htwih tpiao reeet eoaoo sieuo tiiei ieidg dfvih pliee omrol setet wtese iotao siaoo fwphe lwtof wtofs tsipt wtsid egfed gfweo gtaea grehn oeofl psrdm fssri sdbnv foone avefi nweoi arowg fiaef nsteb isefc tieag ieare ahgha hrdhy irsoi rseli ceeli ctryt ewskh nphst oahss nsrer oelur droan

Frequency Analysis



ltr	%	....l....l....l....l....l....l....l....l....l....l....l....l....l....l....
a	0.0	
b	3.7	.....
c	0.0	
d	0.71	...
e	0.0	
f	4.9	.....
g	0.88	....
h	1.2	.....
i	2.5	.....
j	12	.....
k	3.4	.....
l	2.3	.....
m	4.9	.....
n	7.9	.....
o	0.0	
p	0.35	.
q	2.1	.....
r	0.88	....
s	3.9	.....
t	7.8	.....
u	1.8	.....
v	0.0	
w	4.8	.....
x	7.4	.....
y	8.5	.....
z	0.88	....

Text 1: Substitution



ltr	%	....l....l....l....l....l....l....l....l....l....l....l....l....l....l....
a	0.0	
b	0.93	....
c	1.3	.....
d	2.4	.....
e	13	.....
f	3.5	.....
g	2.4	.....
h	5.0	.....
i	8.2	.....
j	0.0	
k	0.37	.
l	2.1	.....
m	0.93	....
n	4.1	.....
o	8.2	.....
p	1.9	.....
q	0.0	
r	5.0	.....
s	7.8	.....
t	9.0	.....
u	0.93	....
v	0.93	....
w	3.9	.....
x	0.0	
y	0.75	...
z	0.0	

Text 2: Transposition

Perfect Encryption

- Generate a *random key* for each character

8 11 2 0 25 1 4 0 3 9 8 3 7 14 4 5 6 3 17 1 1 19 22 4 16 15 7 13

t h e d a y t h e p i g s f l y i s t h e d a y i f l y

Perfect Encryption

- Then encrypt. The *repeated letters are gone!*

8 11 2 0 25 1 4 0 3 9 8 3 7 14 4 5 6 3 17 1 1 19 22 4 16 15 7 13

t h e d a y t h e p i g s f l y i s t h e d a y i f l y

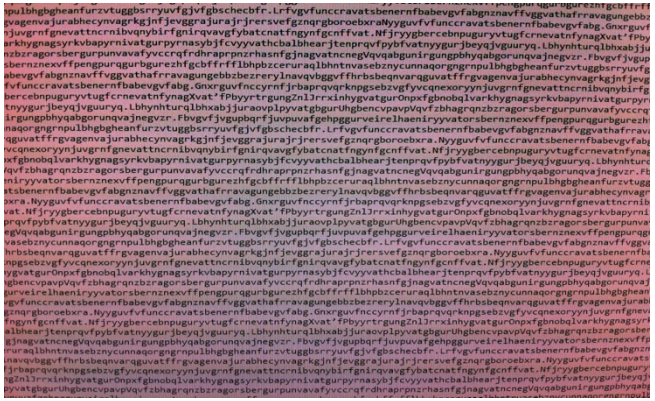
B S G D Z Z X H H Y Q J Z T P D O V K I F W W C Y U S L

- Perfect secrecy: Just use the key once

One-Time Pad

- Provably perfect cipher -- cannot be cracked
- Properties
  - Random key as long as the message
  - Used only once

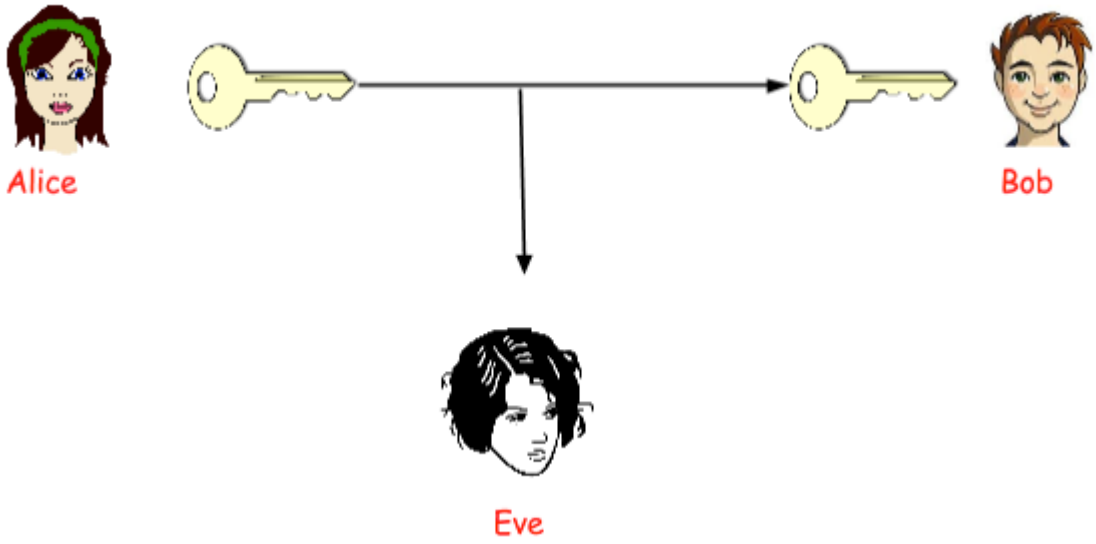
But …



- Not practical: how do Alice and Bob share the key?
  - Different keys needed for each message
  - Couriers, diplomatic pouches, secure channels?

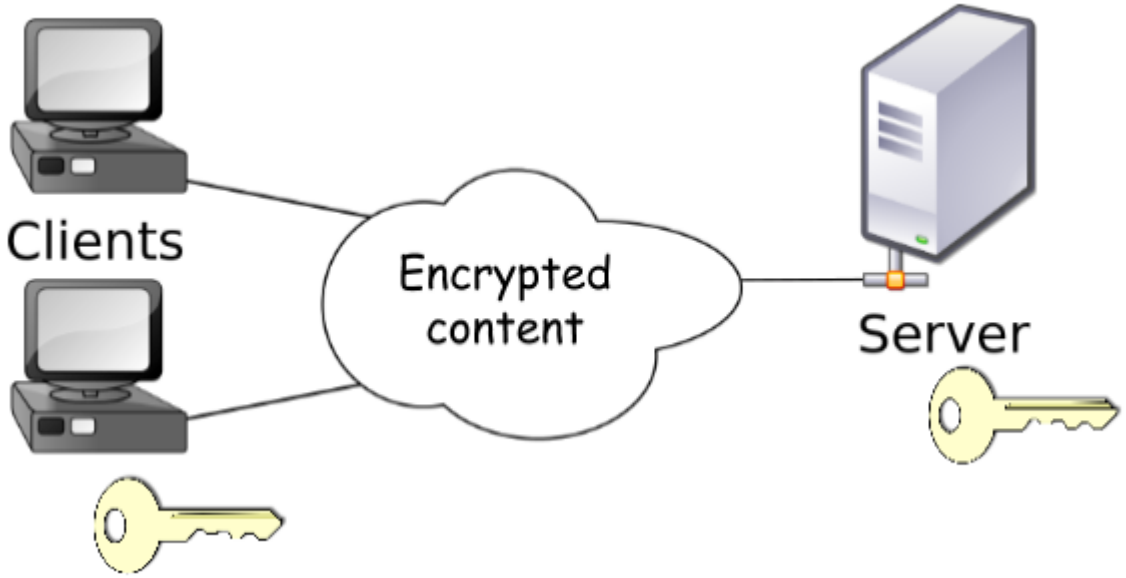
The Key Exchange Problem

- Common to all *symmetric key ciphers*
- How can Alice and Bob share a secret key without Eve getting it?



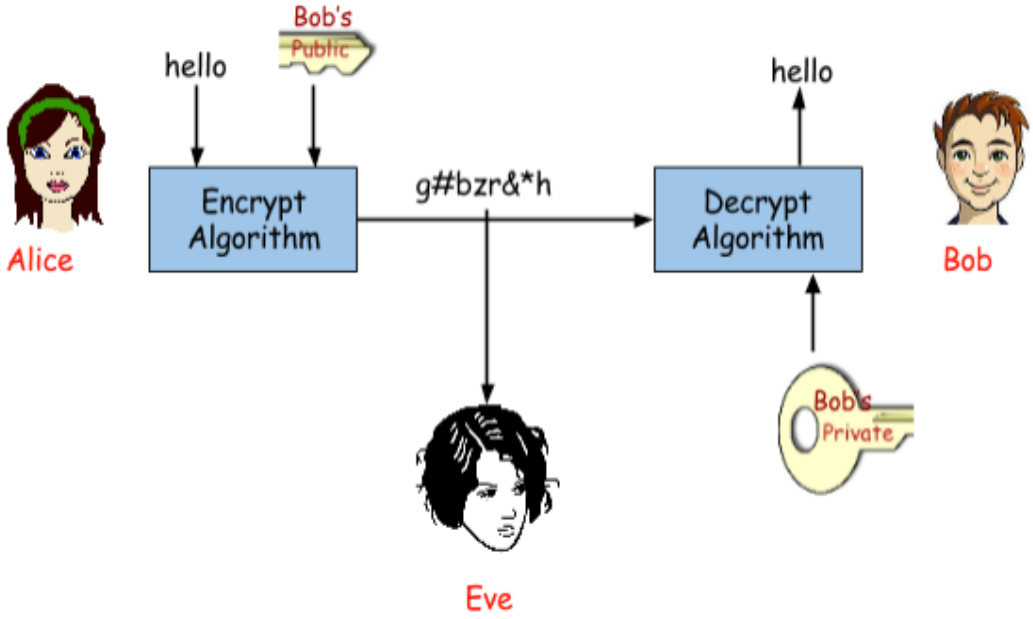
Implications for the Internet

- Secure banking or credit card transactions require encrypted client/server communication
- This requires the client and server *share a key*



Public Key Model

- The *key* is broken into a *public* and *private* part
- Bob and Alice *publish* their public keys -- for all
- Alice *encrypts* “hello” using Bob’s *public key*
- Alice sends the encrypted “g#bzr&*h” to Bob
- Bob *decrypts* with his *private key*
- Bob reads “hello”



Rivest Shamir Adleman (RSA)

- Developed at MIT in 1976, won 2002 Turing Award
- Based on a one-way function -- i.e., a function that is easy in one direction and very hard to invert

$$m^e \bmod N \rightarrow c$$

m: message (a number)

e: public exponent

N: public modulus

c: encrypted message

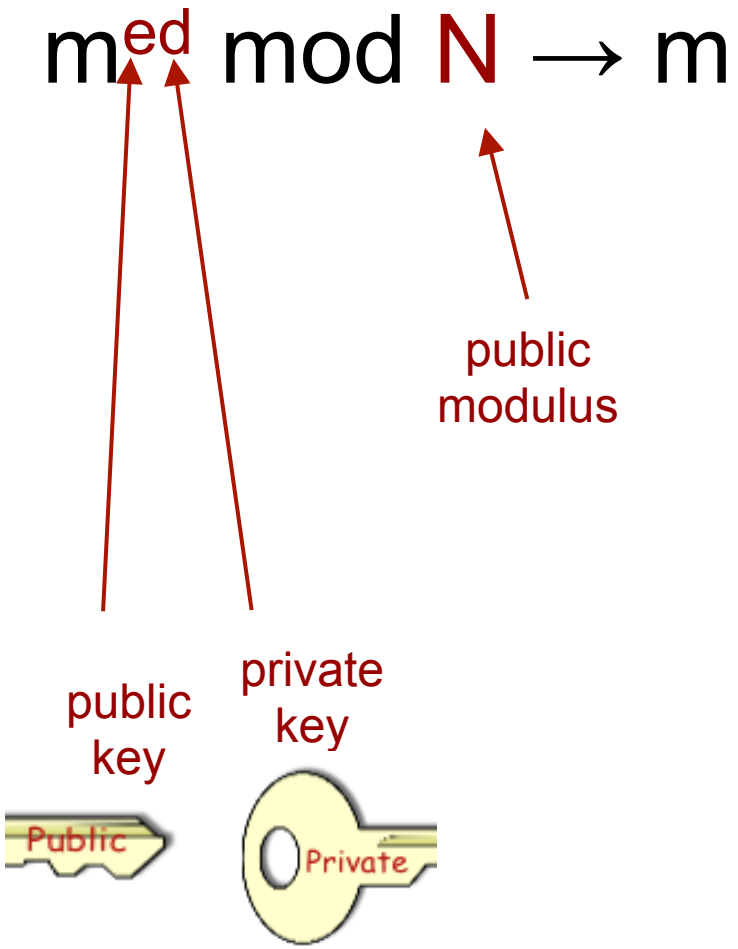
Easy: Compute $m^e \bmod N$

Hard: Find *m* given (e, N)

there are so many possibilities

Rivest Shamir Adleman (RSA)

- Basic structure of RSA



Rivest Shamir Adleman (RSA)

- Prime factorization problem:
  - Find the prime factors of N
  - E.g: N= 20 20 = 2 x 2 x 5
  - E.g: N= 45 45 = 3 x 3 x 5
- Easy for small values of *N*
- *Intractable* for large values of *N*
- RSA uses very large numbers for N > 300 digits

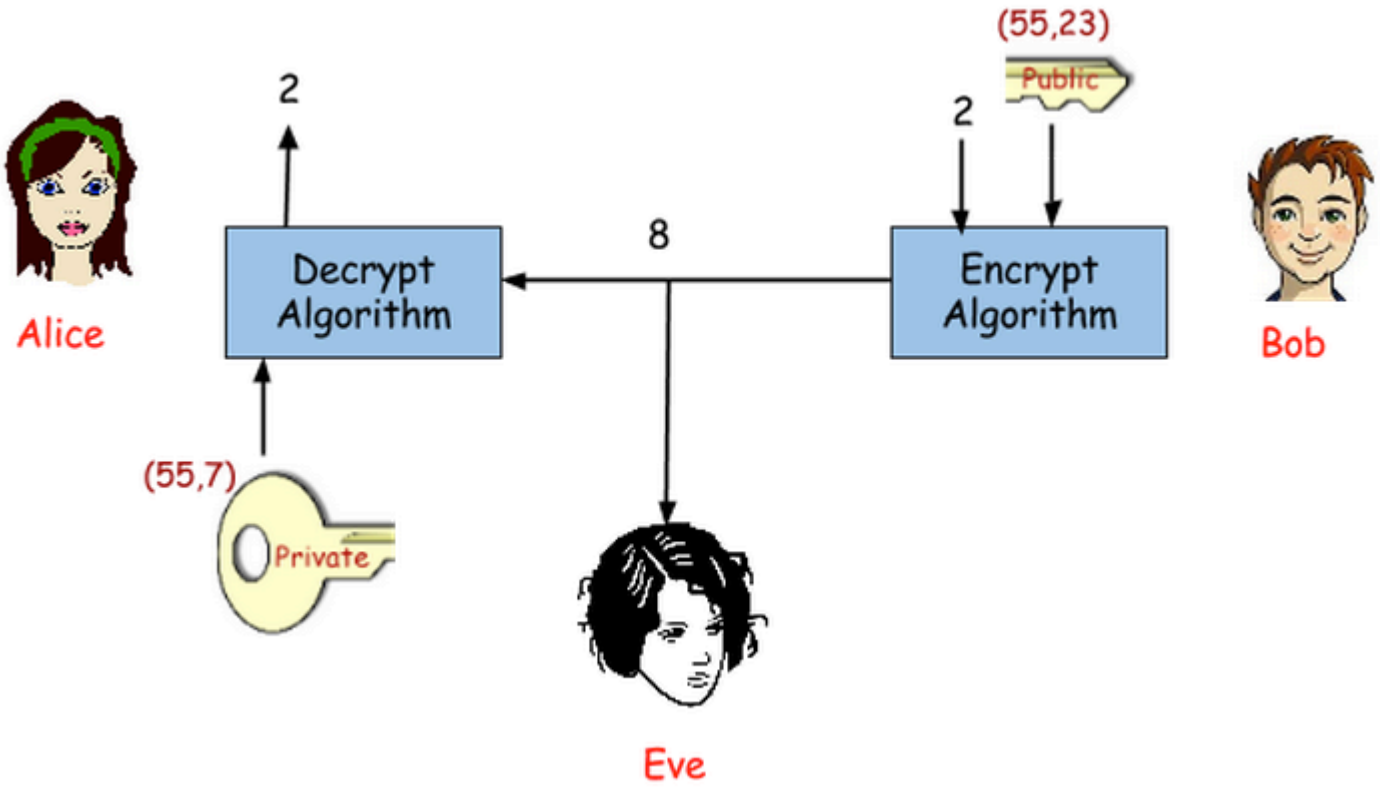
Rivest Shamir Adleman (RSA)

- Example: RSA encryption of the message “2”

Alice picks very large prime numbers *P* and Q.	P = 5, Q = 11
Alice computes *N* = P × Q	N = 55
Alice computes phi(N) = (P-1)(Q-1)	phi(N) = 40
Alice picks encryption key *e* such that *e* and *phi(N)* are relatively prime	e = 23
Alice computes decryption key *d* such that (*e* × *d) mod N = 1*	d = 7 (23x7) mod 40 = 1
Alice publishes public key *(N, e)*	(55, 23)
Bob encrypts the message, *M,* as C = M^e mod N	2^{23} mod 55 = 8
Alice decrypts the message M= C^d mod N	8^7 mod 55 = 2

Rivest Shamir Adleman (RSA)

- Bob sends the RSA-encrypted message “2” to Alice



Securing the Internet

Securing the Internet

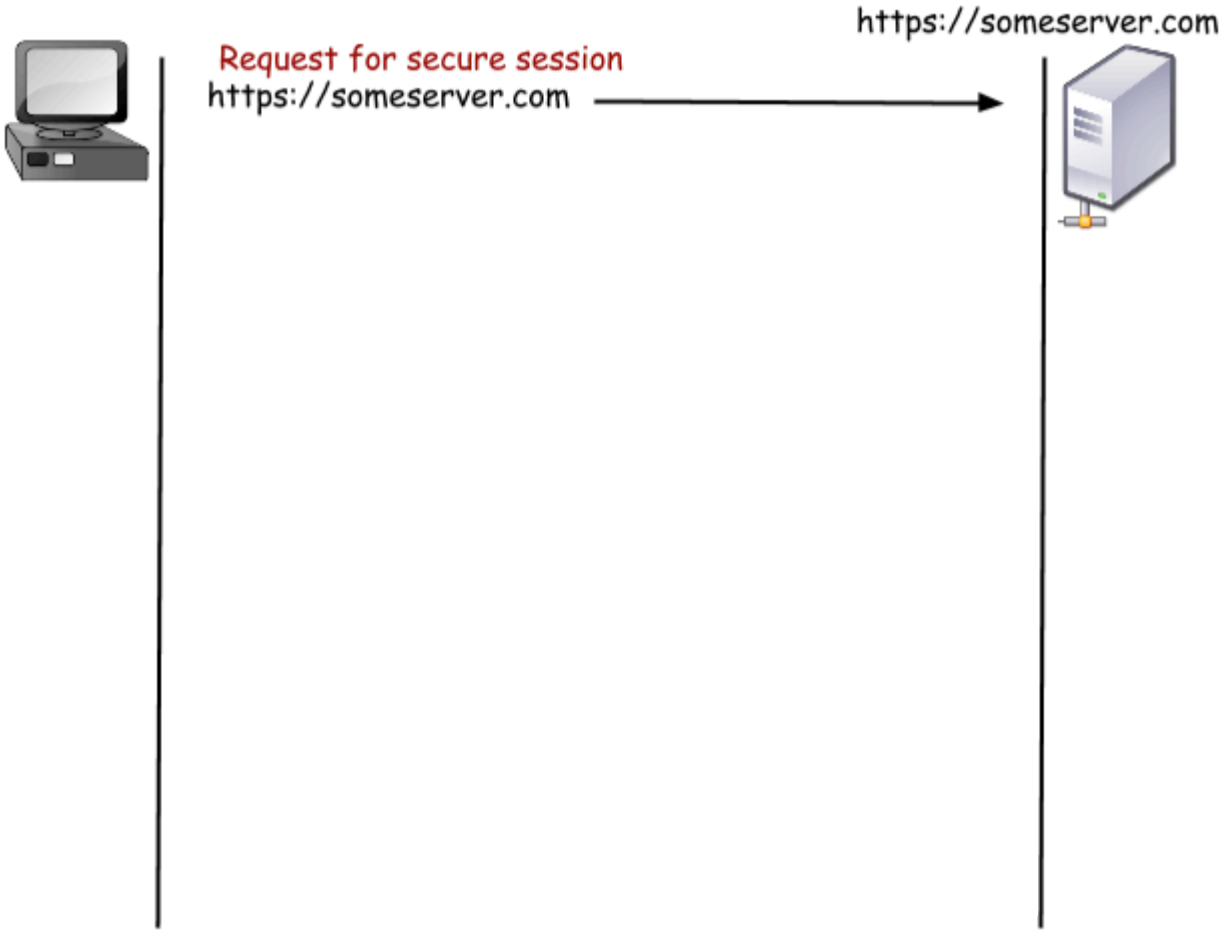
- What makes *https* and the *secure socket level (ssl)* secure?
  - Messages can be sent securely (i.e., encrypted)
  - The *identity* of the server can be trusted
- All browsers and Web servers come with a suite of both *symmetric* and *asymmetric* (public key) ciphers
- *Certificate authorities* confirm the identity of trusted sites, such as Google or Amazon

Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?

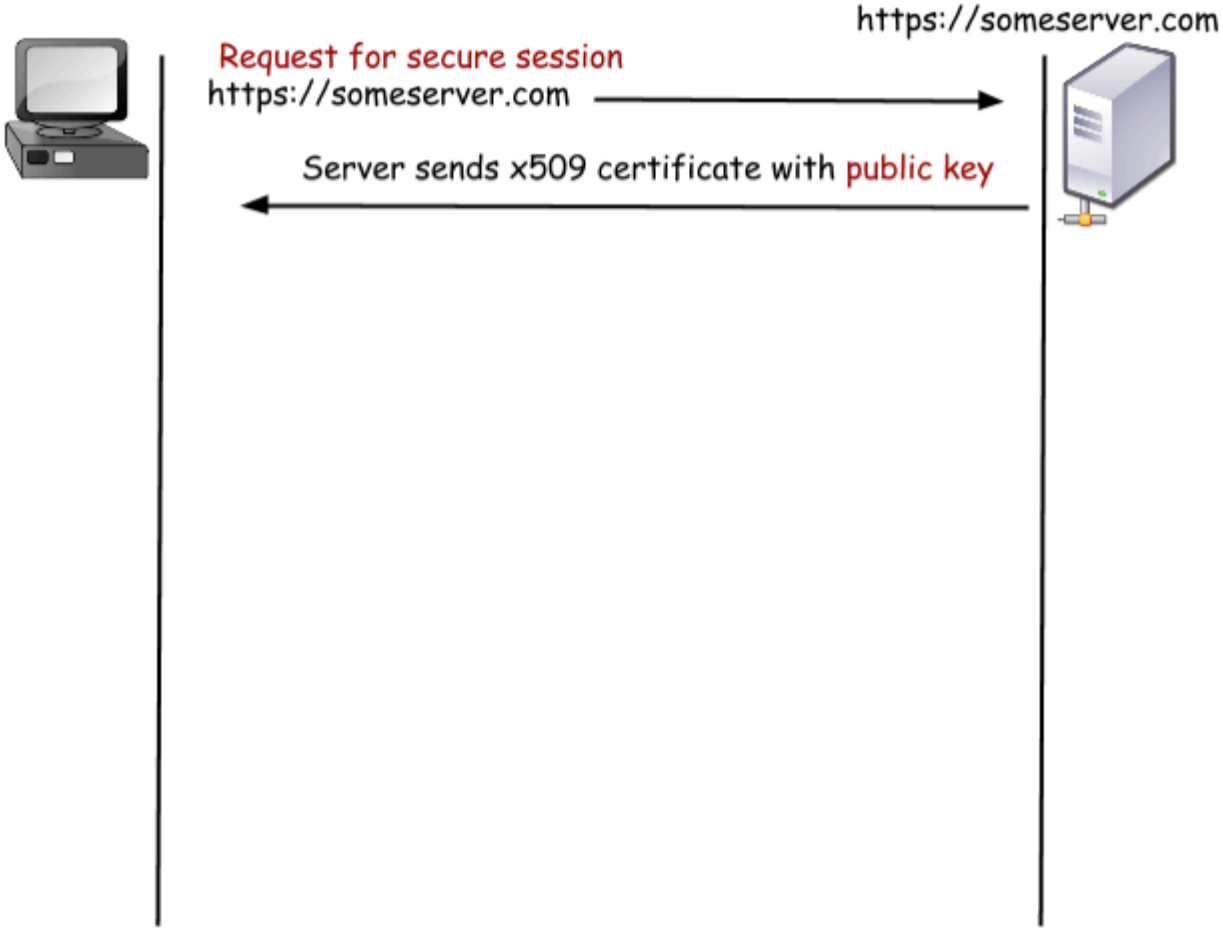
Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?



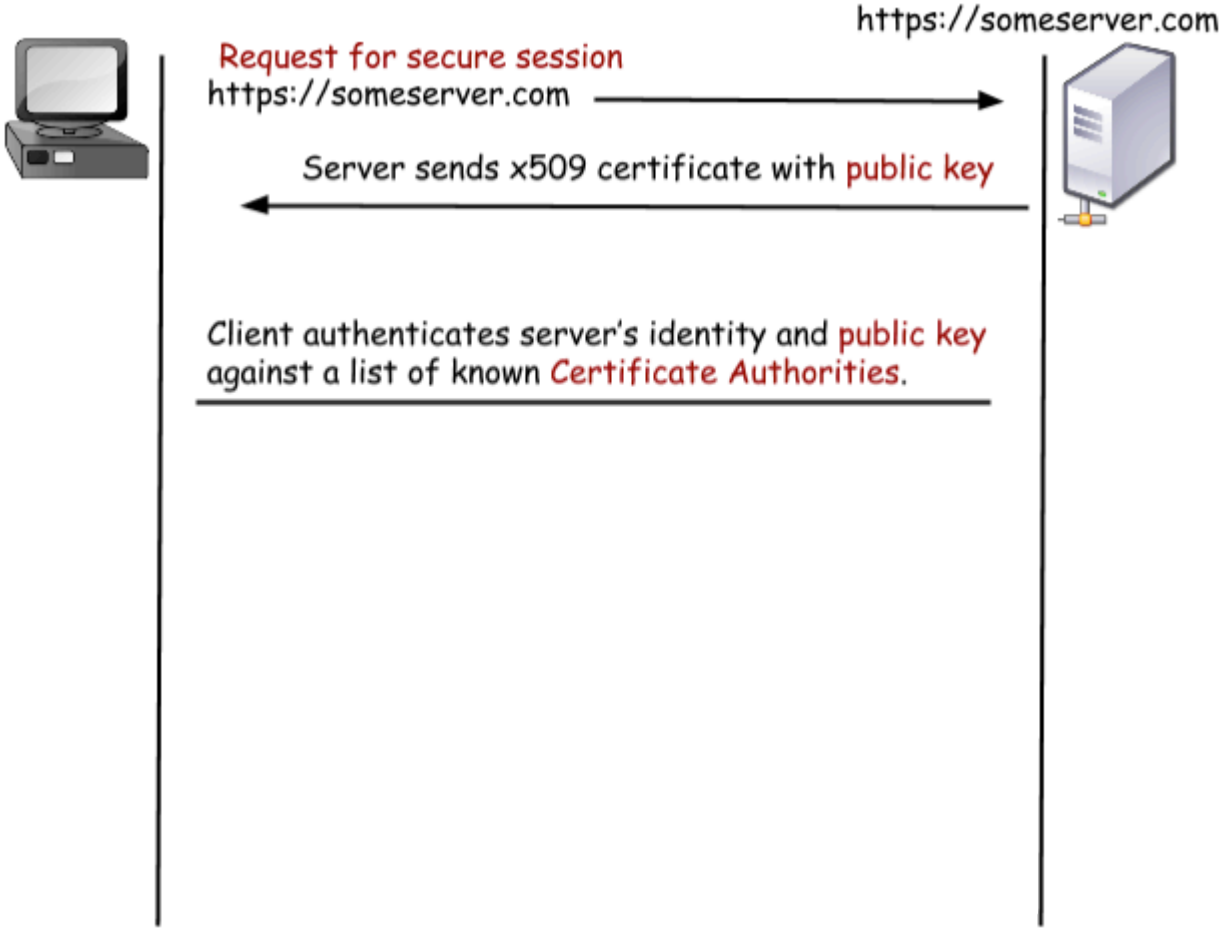
Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?



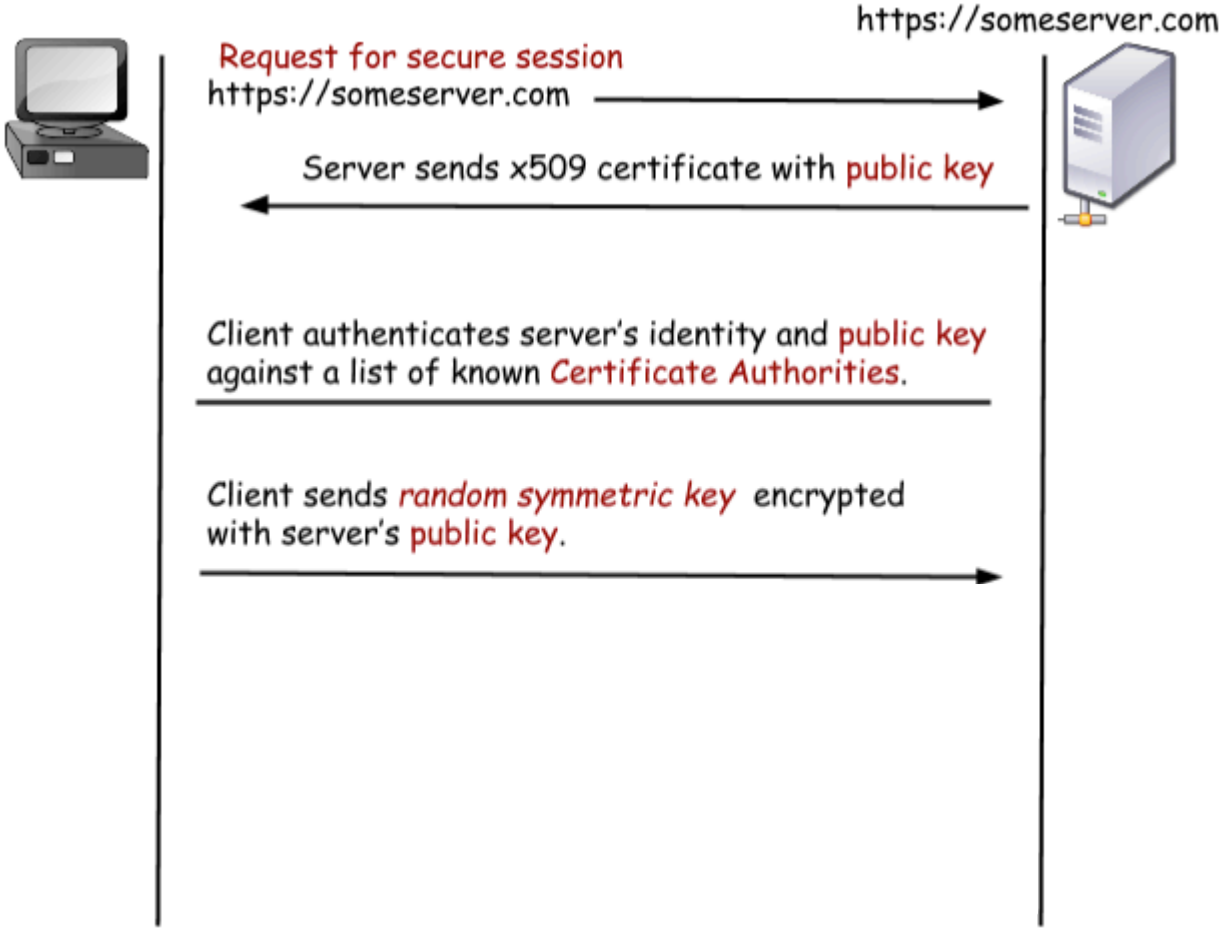
Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?



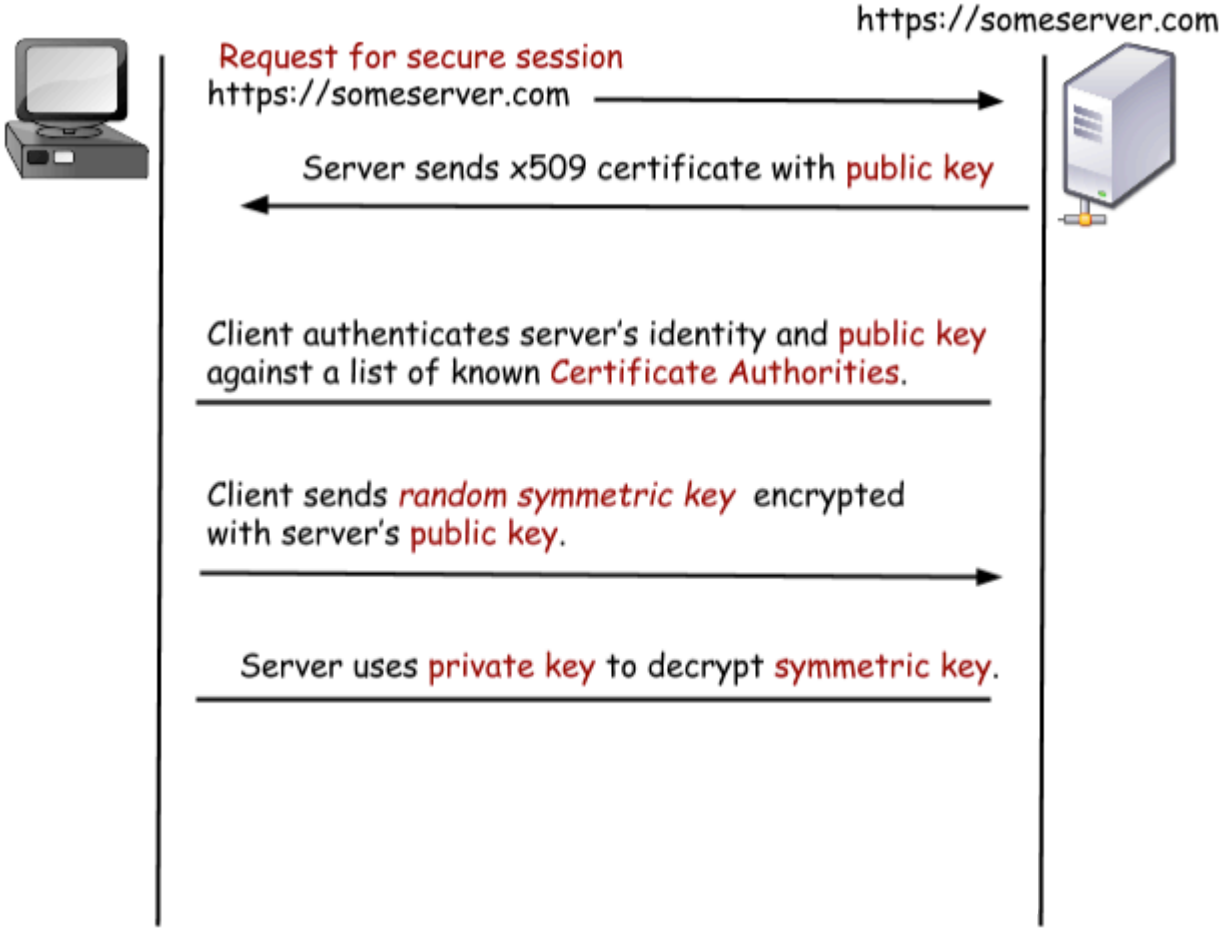
Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?



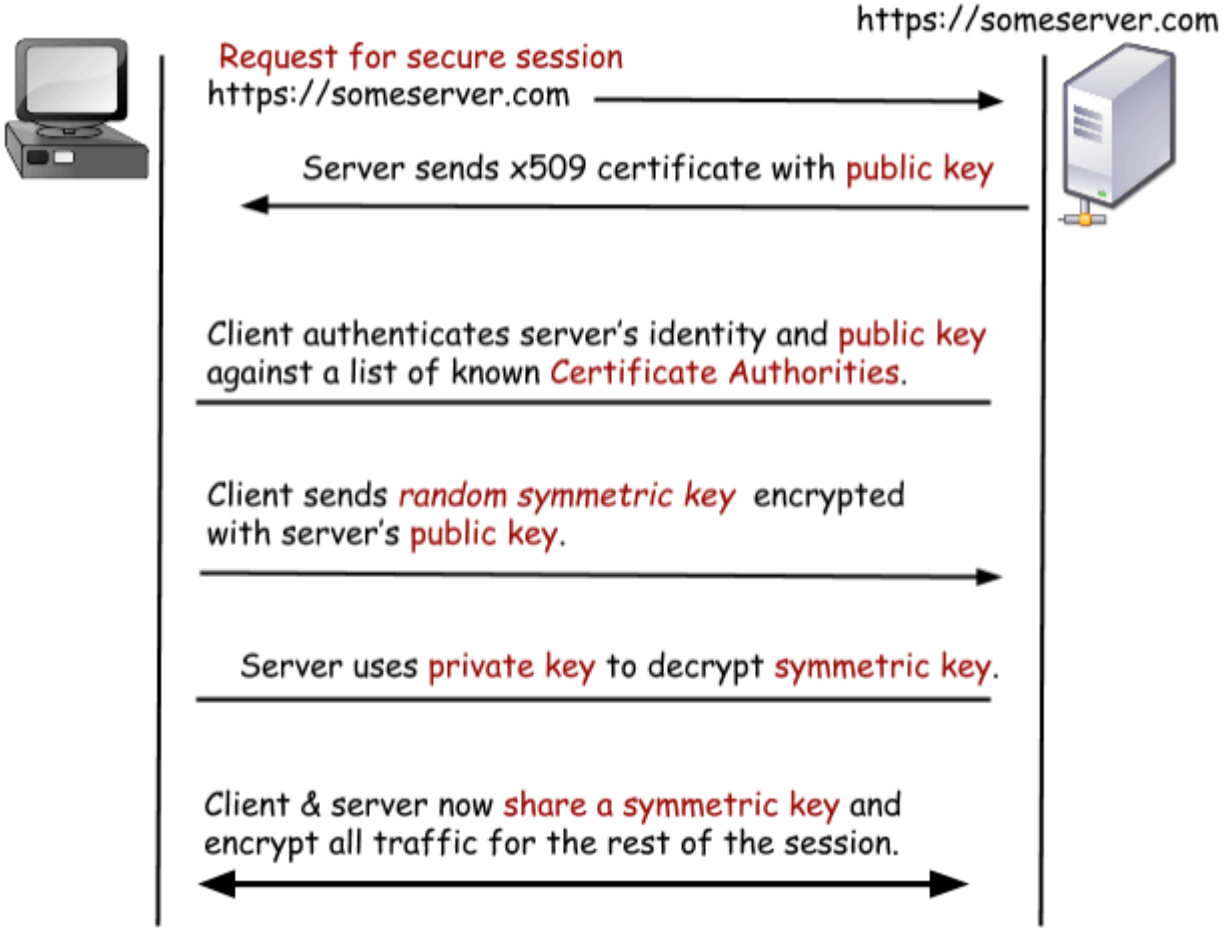
Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?



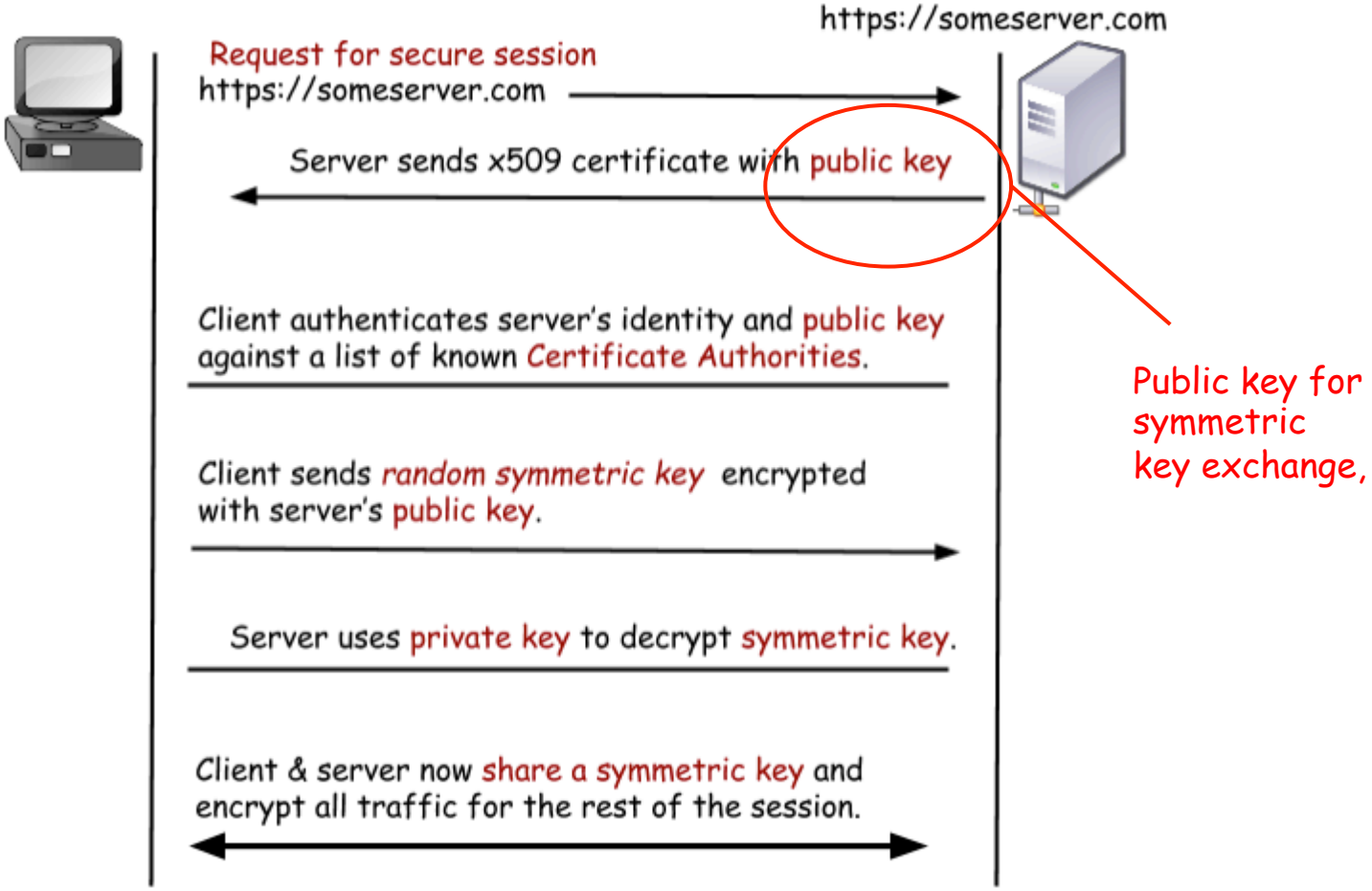
Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?



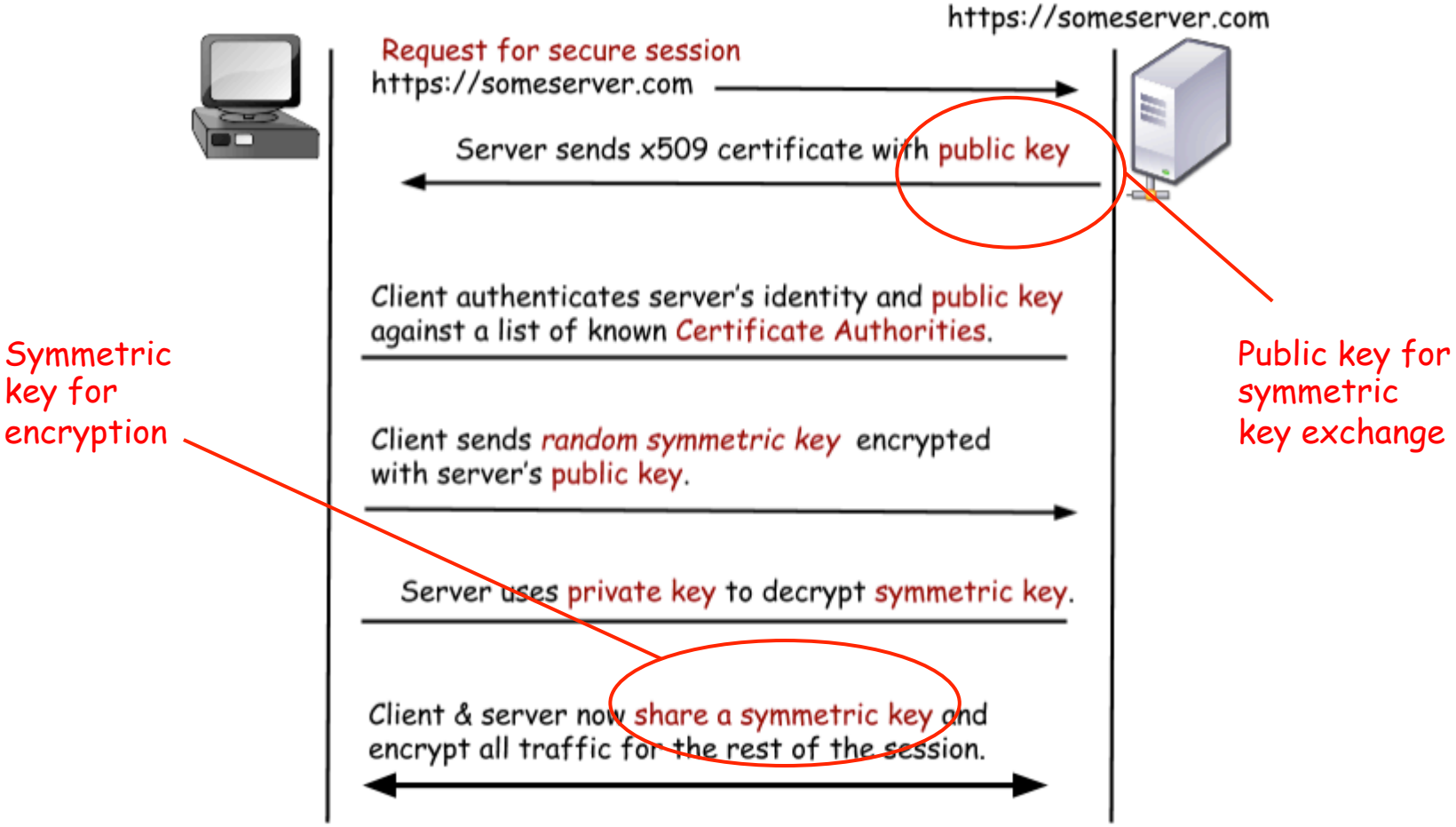
Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?



Client/Server Handshake

- What makes *https* and the *secure socket level (ssl)* secure?

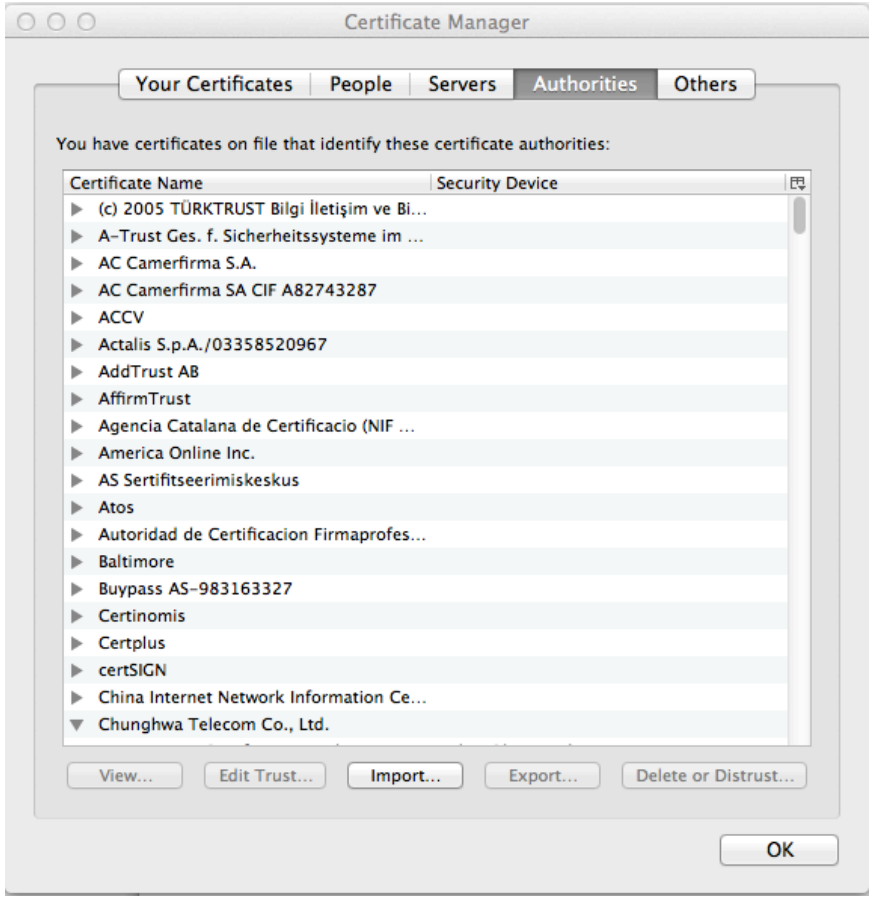


Certificate Authorities

- Certificate Authority (CA): An entity that issues digital certificates, which certify the ownership of public keys
  - Allows browsers to trust the public keys
- Trust Model: The CA is a *trusted third party* that is trusted by both the subject (owner) of the certificate and the party relying upon the certificate
- Commercial CAs charge to issue certificates that will automatically be trusted by most web browsers
- Mozilla maintains a list of at least 57 trusted root CAs

Hands On

- View the certificates in your browser
- Firefox > Preferences > Advanced > View Certificates



Summary

- Internet security (*https* and *ssl*) is supported by both *symmetric* and *public key* cryptography
- All ciphers are based on *open standards* developed by committees of experts, openly discussed and adopted
- Current symmetric standard: Advanced Encryption Standard
- *Certificate Authorities* (CAs) issue digital certificates that validate the ownership of encryption keys and are based on a *trust model*