

# CSCI 476: Computer Security

Secret Key Encryption/Symmetric Cryptography (Part 1)

Reese Pearsall  
Fall 2022

# Announcement

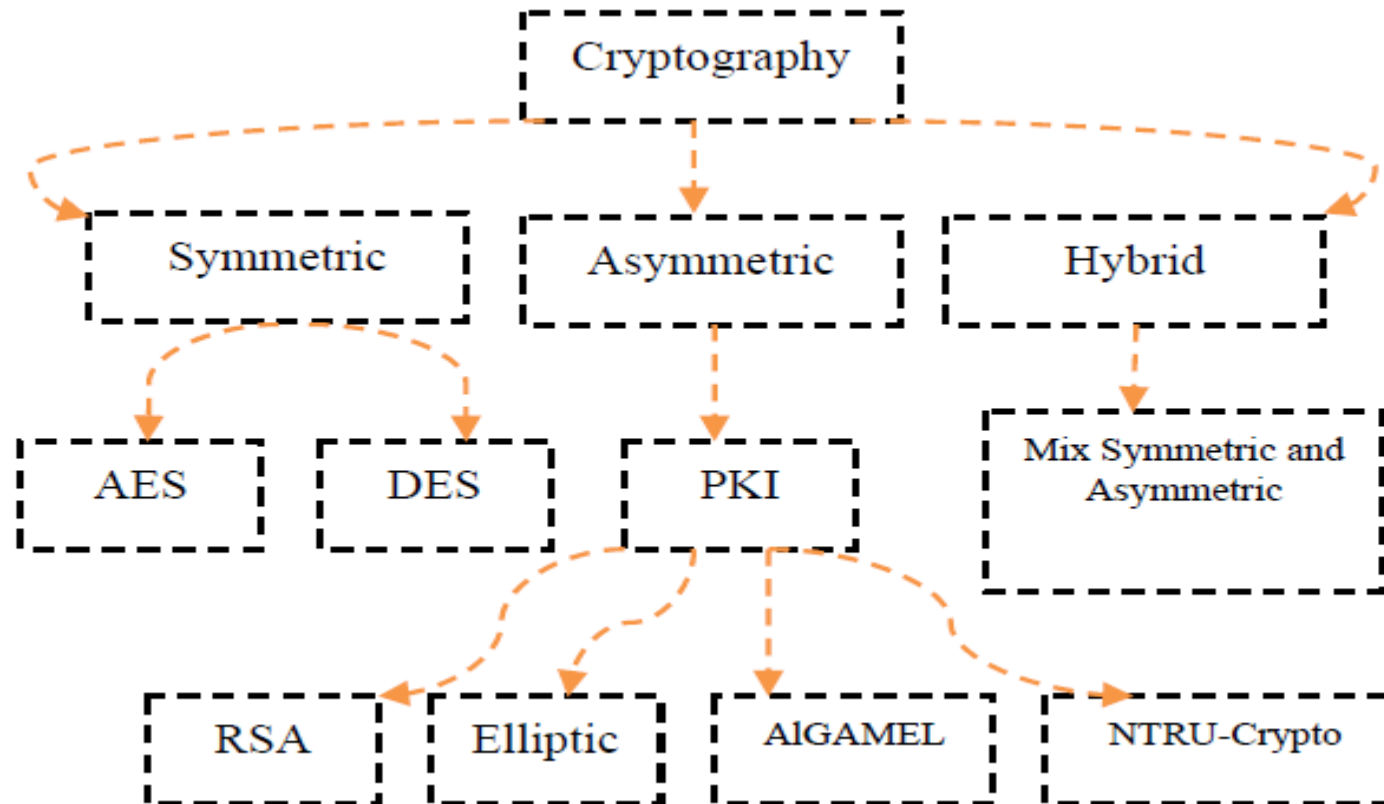
TCP/IP Lab due tonight!!

Project due one week from  
today (11/16)

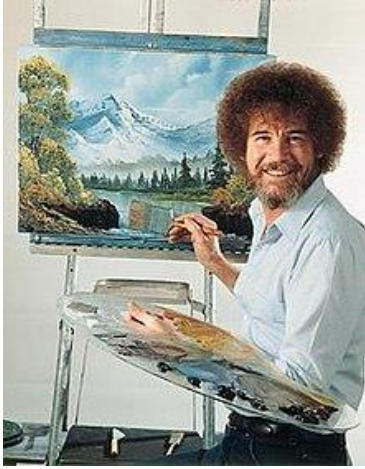
DNS Lab due next Sunday  
(11/19)

- The protection of information and information systems

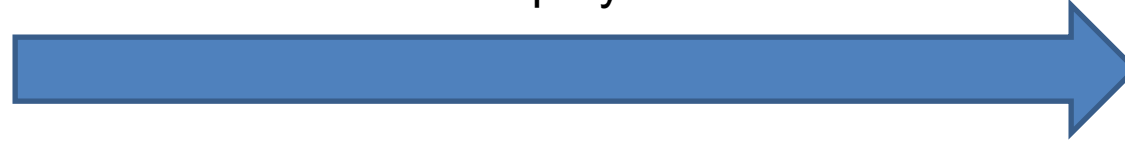
**Cryptography** is the practice and study of techniques for securing communications and data in the presence of adversaries



Bob



→ “Hi Alice, my address is  
123 Painting Avenue.  
Please stop by at 6:00” →

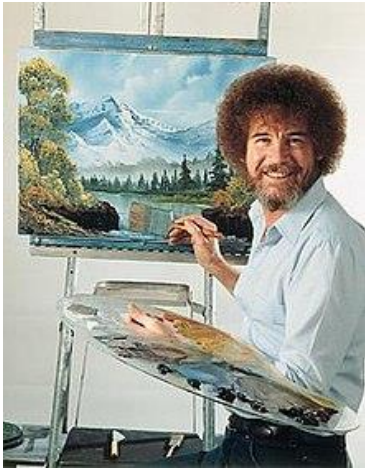


Over a wire, wirelessly, via a Pidgeon etc

Alice



Bob

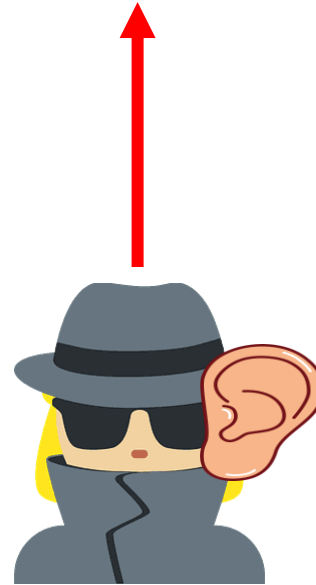


“Hi Alice, my address is  
123 Painting Avenue.  
Please stop by at 6:00”

Alice

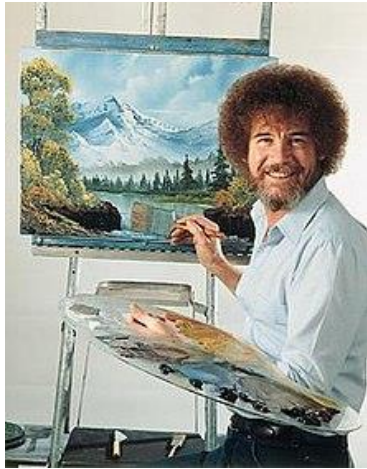


Because our transmission medium is  
**shared**, there is a possible someone  
else could be eavesdropping



Eve

Bob

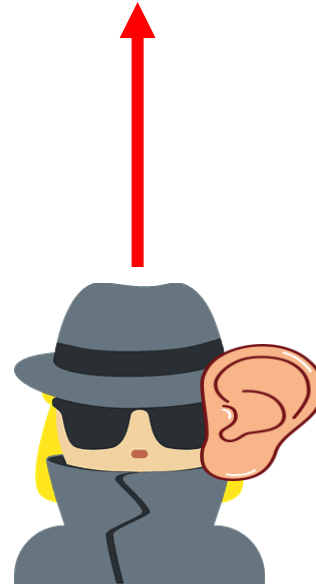


“Hi Alice, my address is  
123 Painting Avenue.  
Please stop by at 6:00”

Alice



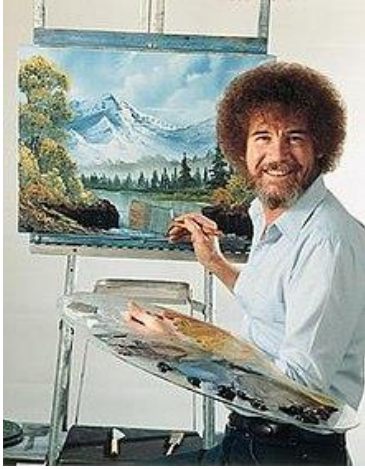
Because our transmission medium is  
**shared**, there is a possible someone  
else could be eavesdropping



Eve

Our goal is to make sure Alice can receive our message securely,  
and our original message cannot be intercepted

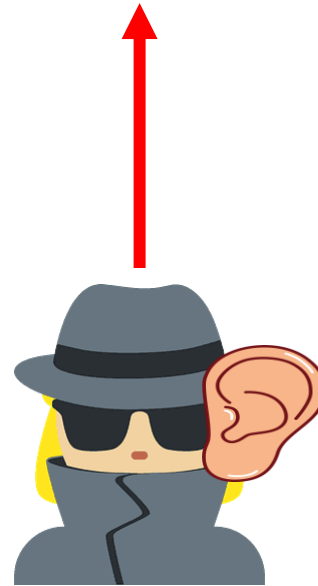
Bob



### Cleartext/Plaintext

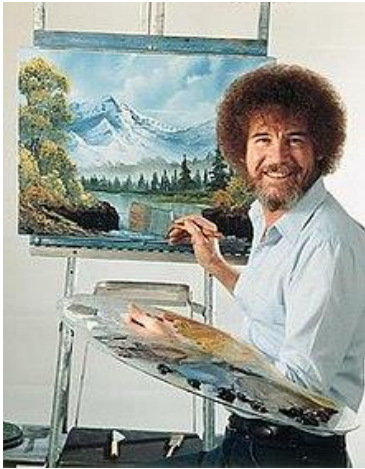
“Hi Alice, my address is  
123 Painting Avenue.  
Please stop by at 6:00”

Alice



Eve

Bob



### Cleartext/Plaintext

“Hi Alice, my address is  
123 Painting Avenue.  
Please stop by at 6:00”

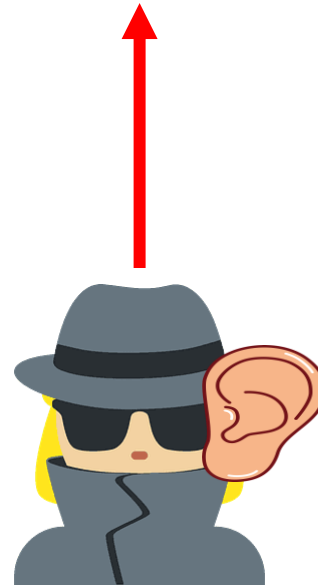


Bob **encrypts** his message with a **key**

MuYGoP5LiTTGPVX6U/r2VTpxPSqT  
Fmy5nsoFWURThKMhHk/7tbjYsS2EJ  
917q7megTAcV+V4ZMU4HjJjiW2DC  
BroxvJ0V3ZYDgZ8B9IUvGUmdiRMH  
25Xkf7QrhAGR3FF

The result is a  
**ciphertext**

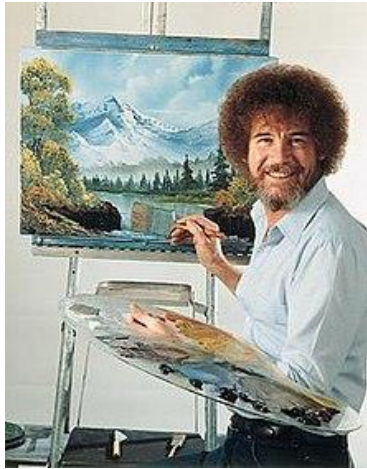
Alice



Eve



Bob



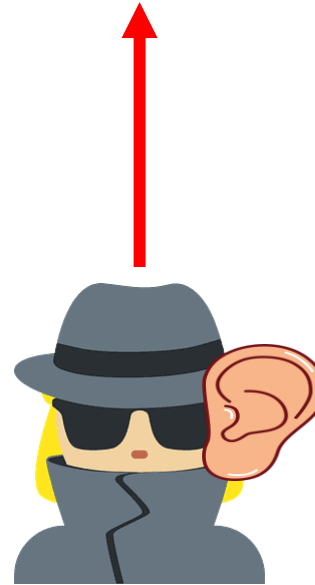
MuYGoP5LiTTGPVX6U/r2VTpxPSqTFmy5nsoF  
WURThKMhHk/7tbjYsS2EJ917q7megTAcV+V4Z  
MU4HjJjiW2DCBroxvJ0V3ZYDgZ8B9IUvGUmdiR  
MH25Xkf7QrhAGR3FF

Alice



## Cleartext/Plaintext

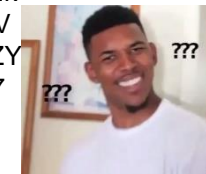
“Hi Alice, my address is  
123 Painting Avenue.  
Please stop by at 6:00”



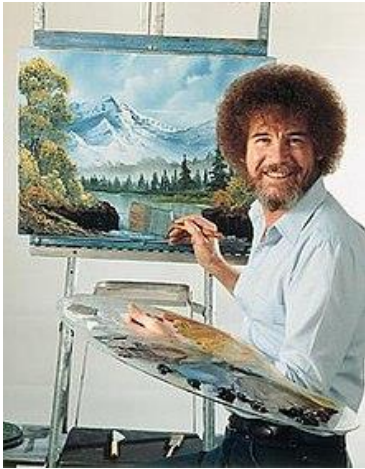
Eve

If Eve intercepts our ciphertext,  
she can't do very much with it

MuYGoP5LiTTGPVX6U/r2VTpx  
PSqTFmy5nsoFWURThKMhHk  
/7tbjYsS2EJ917q7megTAcV+V  
4ZMU4HjJjiW2DCBroxvJ0V3ZY  
DgZ8B9IUvGUmdiRMH25Xkf7  
QrhAGR3FF



Bob



### Cleartext/Plaintext

“Hi Alice, my address is  
123 Painting Avenue.  
Please stop by at 6:00”



Eve

Alice receives the  
ciphertext, and then uses  
the **same key** that bob  
used, and then **decrypts**  
the ciphertext

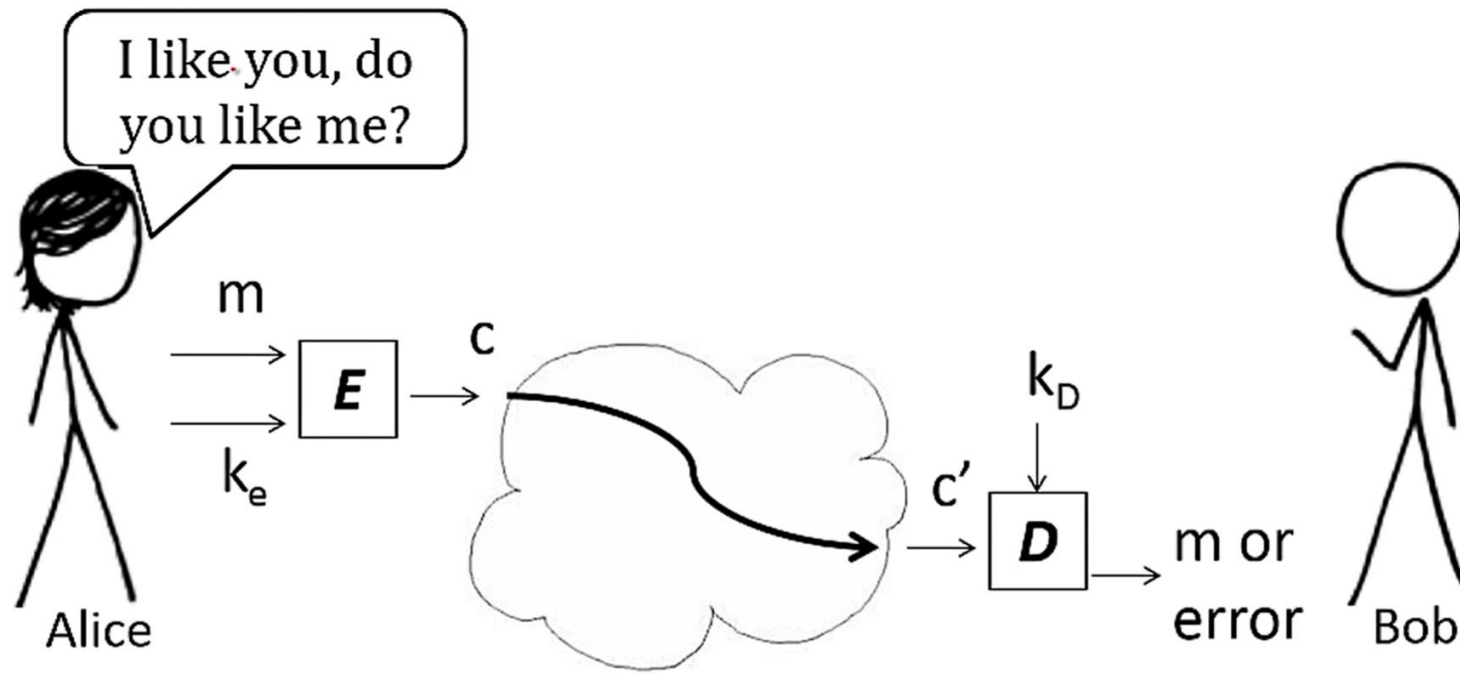
Alice



MuYGoP5LiTTGPVX6U/r2  
VTpxPSqTFmy5nsoFWUR  
ThKMhHk/7tbjYsS2EJ917  
q7megTAcV+V4ZMU4HjJji  
W2DCBroxvJ0V3ZYDgZ8  
B9IUvGUmdiRMH25Xkf7  
QrhAGR3FF



“Hi Alice, my address is  
123 Painting Avenue.  
Please stop by at 6:00”



## Cryptosystem

$m$ : Plaintext

$c$ : Ciphertext

$k_e$ : Encryption Key

$E$ : Encryption Program

$k_d$ : Decryption Key

$D$ : Decryption Program

Deterministic programs\*

The importance here is that the **keys** used for encryption/decryption are secret (ie not public knowledge)

The innerworkings of the encryption/decryption program *is* public knowledge though

Secure cryptography is the foundation for our secure communications in the cyber world (HTTPS, SSH, etc)

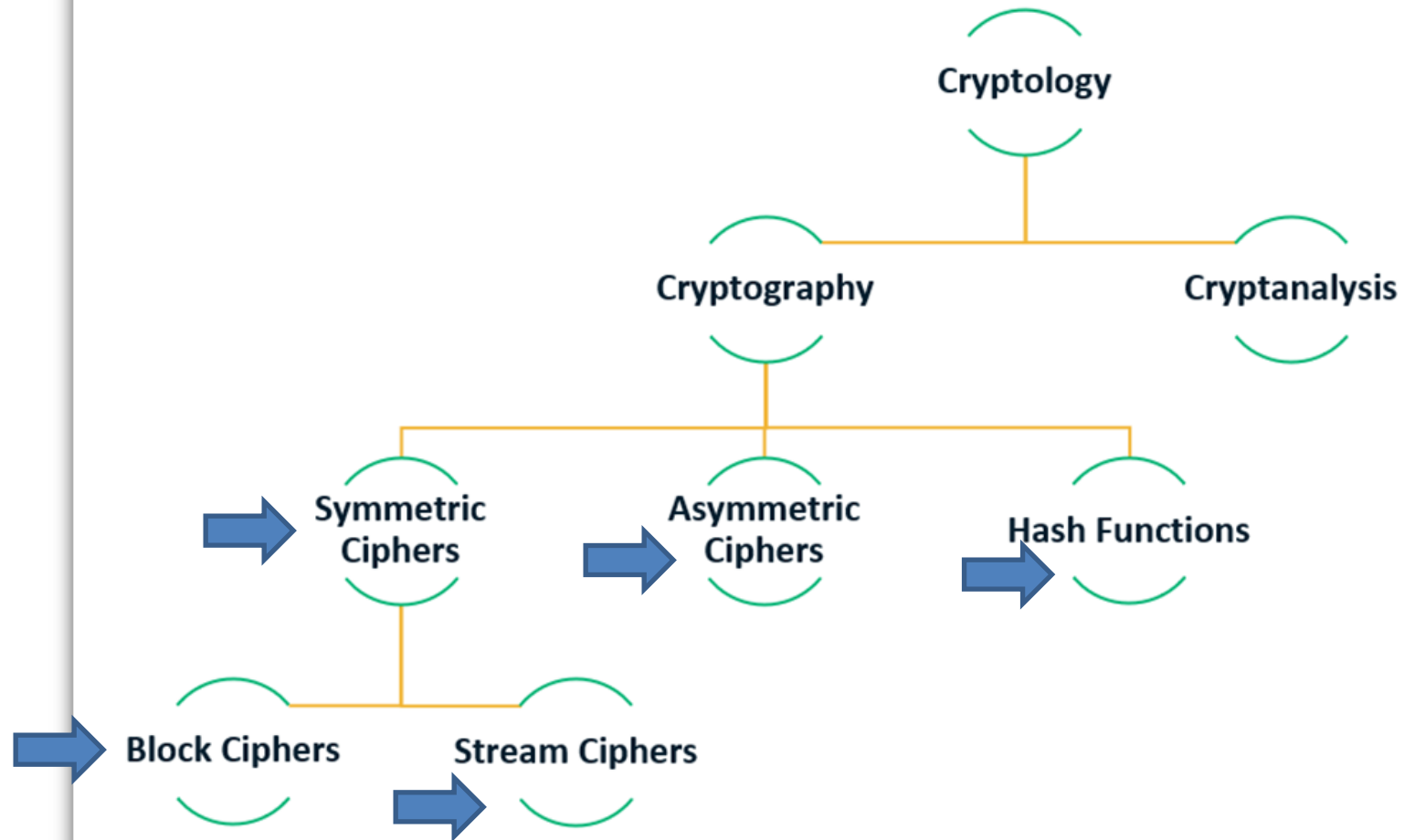
The encryption algorithms are typically rooted in **very difficult problems** in computing (ie there does not exist a program that can efficiently break RSA **YET**)

There are very intense proofs and prove the secureness of the encryption procedures we use today

Never try to roll out your own cryptography scheme, and never use the built-in RNG for secure communications (import random)



# Cryptology vs Cryptography



# Early cryptography techniques

**Caesar Cipher-** letters in the plaintext will be replaced by some fixed number of positions down in the alphabet.

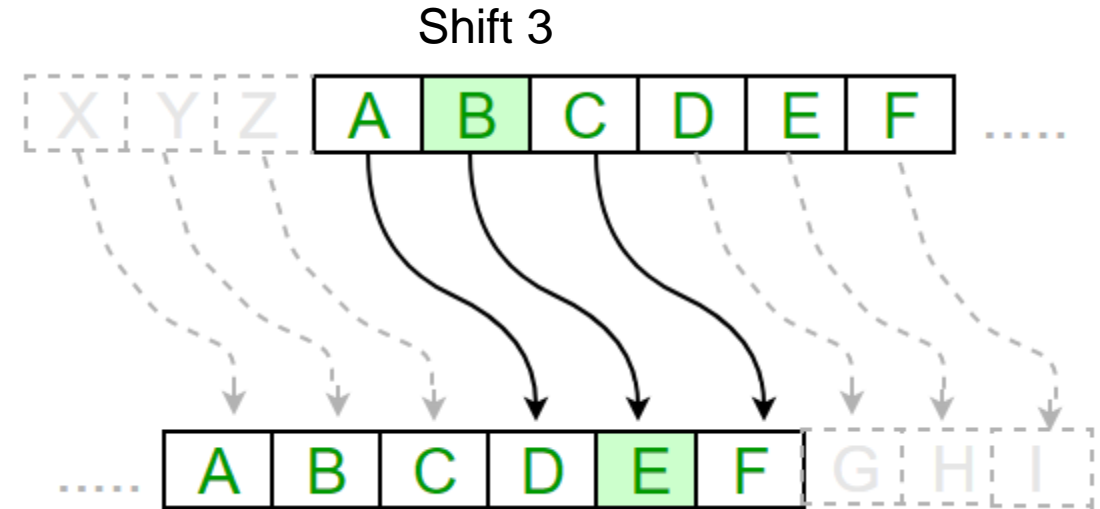
hello there world my name is reese



khoor wkhuh zruog pb  
qdph 1v uhhvh

plaintext

ciphertext



Nifty, but we have the technology to brute force 26 possible shifts

# Substitution Cipher

Letters in plaintext are substituted by another letter

E → X

R → Z

REESE = ZXXSX

**Monolithic Substitution Cipher** – Same “rules” are applied throughout the entire plaintext

**Polyalphabetic Substitution Cipher** – different “rules” are applied throughout the plaintext

open 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  
KEYWORD A B C F G H I J L M N P Q S T U V X Z  
cipher alphabet

keyword: KEYWORD  
plain text: A L K I N D I  
ciphertext: K

Here is a ciphertext (cipher.txt)

```
ydq ufyiqoobxrk lrcqx yqoy fo r kwgyfoyrbq rxxepfc crlrcfyt yqoy ydry lxebxqoofvqgt bqyo kexq  
mfuufcwgy ro fy ceiyfiwqo. ydq ysqiylt kqyqx lrcqx yqoy sfgg pqbfi fi ydfxyt oqceimo. gfiq wl ry ydq  
oyrxy. ydq xwiifib olqqm oyrxyo ogesgt, pwy bqyo uroyqx qrcd kfiwyq ruyqx tew dqr x ydfo ofbirg pql r  
ofibgq grl odewgm pq ceklgqyqm qrcd yfkq tew dqr x ydfo oewim. [mfib] xqkqkpqx ye xwi fi r oyxrfdy  
gfiq, rim xwi ro geib ro leoofpgq. ydq oqceim yfkq tew urfg ye ceklgqyq r grl pquexq ydq oewim, tew  
yqoy fo evqx. ydq yqoy sfgg pqbfi ei ydq sexm oyrxy. ei tewx krxj, bqy xqrmt, oyrxy.
```

Suppose we know that that this message is an english message encrypted with a monolithic substitution cipher

Can we crack this?



Here is a ciphertext (cipher.txt)

```
ydq ufyiqoobxrk lrcqx yqoy fo r kwgyfoyrbq rxxepfc crlrcfyt yqoy ydry lxxbxqoofvqgt bqyo kexq  
mfuufcwgy ro fy ceiyfiwqo. ydq ysqiyl kqyqx lrcqx yqoy sfgg pqbfi fi ydfxyt oqceimo. gfiq wl ry ydq  
oyrxy. ydq xwiifib olqqm oyrxyo ogesgt, pwy bqyo uroyqx qrcd kfiwyq ruyqx tew dqr x ydfo ofbirg pql r  
ofibgq grl odewgm pq ceklgqym qrcd yfkq tew dqr x ydfo oewim. [mfib] xqkqkpx ye xwi fi r oyxrbdy  
gfiq, rim xwi ro geib ro leoofpgq. ydq oqceim yfkq tew urfg ye ceklgqy r grl pquexq ydq oewim, tew  
yqoy fo evqx. ydq yqoy sfgg pqbfi ei ydq sexm oyrxy. ei tew krxj, bq xqrmt, oyrxy.
```

**Frequency Analysis** leverages the fact that in any given written language, certain letters and combinations occur more frequently than others

In English, T, A, I, and O are the most common letters, so it is likely the letters that appear the most frequently in our ciphertext are one of those

```

y dq ufyiqoobxr k lrcqx yqoy fo r kwgyfoyrbq r qxepfc c r lrcfy t yqoy ydry l xebxqoofvqgt b qyo kexq
m fuufcwgy ro fy ceiyfiwqo. y dq ys qiy t k qyqx lrcqx yqoy sfgg p qbf i fi ydfxy t oqceimo. gfiq wl ry ydq
oyrxy. y dq xwiifib olqam oyrxyo ogesgt, pwy b qyo uroyqx qrcd kfiwyq ruyqx tew d qrx ydfo ofbirg p qql r
ofibgq grl odewgm p q ceklqyqm qrcd yfkq tew d qrx ydfo oewim. [mfib] xqkqkpx ye xwi fi r oyxfbdy
gfiq, rim xwi ro geib ro leoofpgq. y dq oqceim yfkq tew urfg ye ceklqyq r grl pquexq y dq oewim, tewx
yqoy fo evqx. y dq yqoy sfgg p qbf i ei y dq sexm oyrxy. ei tewx kr xj, b qy xqrmt, oyrxy.

```

Here is a ciphertext (cipher.txt)

We can write a program that counts the frequency of characters (**1-gram**) and frequency of character pairs (**2-gram**)

```
[11/03/22] seed@VM:~/encryption_lecture$ ./freq.py < cipher.txt
```

-----  
1-gram (top 20):

```

q: 61
y: 58
o: 39
r: 34
f: 32
x: 30
i: 27
e: 26
g: 21
d: 18
w: 17
b: 14
c: 13
k: 12
l: 12
m: 12
t: 11
p: 9
.: 8
u: 7

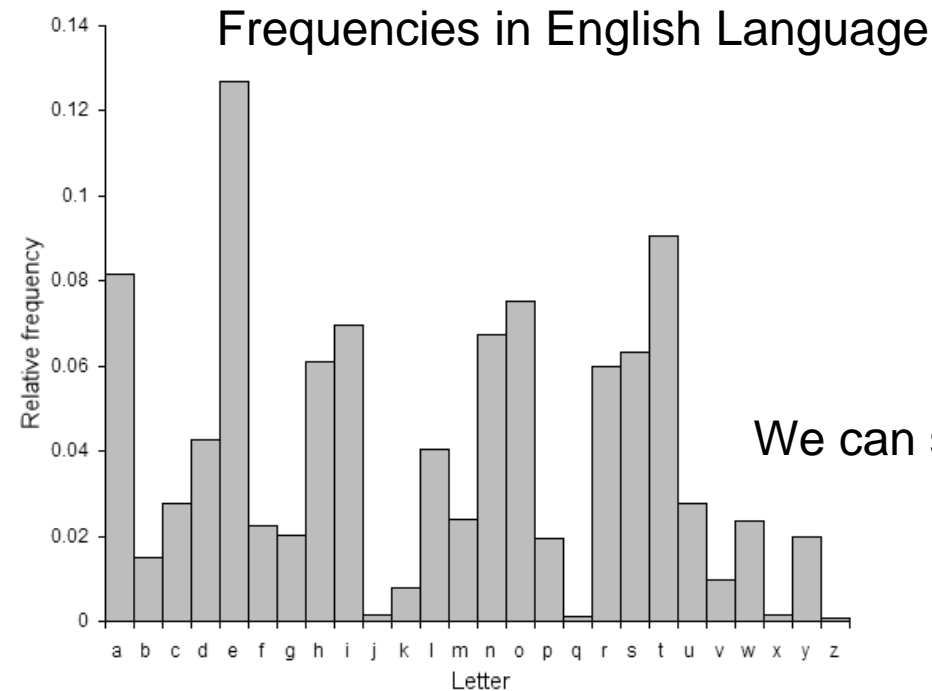
```

2-gram (top 20):

```

yd: 12
oy: 12
yq: 11
fi: 11
dq: 10
qo: 8
qx: 8
ew: 8
rx: 7
qy: 6
ei: 6
pq: 6
rc: 5
fo: 5
yr: 5
xq: 5
ce: 5
xy: 5
im: 5
wi: 5

```



We can start making guesses!

**Most common bigrams (in order)**

th, he, in, en, nt, re, er, an, ti, es, on, at, se, nd, or, ar, al, te, co, de, to, ra, et, ed, it, sa, em, ro.

Here is a ciphertext (cipher.txt)

```
[11/03/22]seed@VM:~/encyption_lecture$ ./freq.py < ciphertext.txt
```

```
q: 61
y: 58
o: 39
r: 34
f: 32
x: 30
i: 27
e: 26
g: 21
d: 18
w: 17
b: 14
c: 13
k: 12
l: 12
m: 12
t: 11
p: 9
.: 8
u: 7
```

```
yd: 12
oy: 12
yq: 11
fi: 11
dq: 10
qo: 8
qx: 8
ew: 8
rx: 7
qy: 6
ei: 6
pq: 6
rc: 5
fo: 5
yr: 5
xq: 5
ce: 5
xy: 5
im: 5
wj: 5
```



th, he, in, en, nt, re, er, an, ti, es, on, at, se, nd, or, ar, al, te, co,  
de, to, ra, et, ed, it, sa, em, ro.

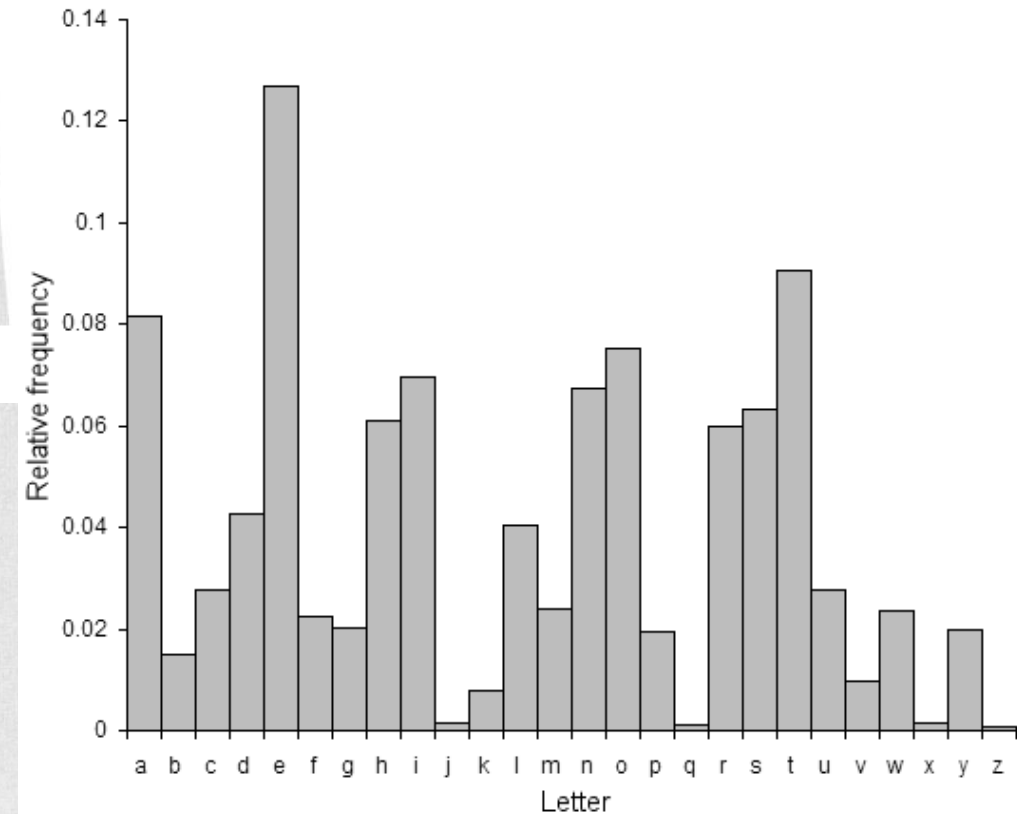
Listing 24.2: Bigram and trigram frequencies

Bigram frequency in English

TH : 2.71	EN : 1.13	NG : 0.89
HE : 2.33	AT : 1.12	AL : 0.88
IN : 2.03	ED : 1.08	IT : 0.88
ER : 1.78	ND : 1.07	AS : 0.87
AN : 1.61	TO : 1.07	IS : 0.86
RE : 1.41	OR : 1.06	HA : 0.83
ES : 1.32	EA : 1.00	ET : 0.76
ON : 1.32	TI : 0.99	SE : 0.73
ST : 1.25	AR : 0.98	OU : 0.72
NT : 1.17	TE : 0.98	OF : 0.71

Trigram frequency in English

THE : 1.81	ERE : 0.31	HES : 0.24
AND : 0.73	TIO : 0.31	VER : 0.24
ING : 0.72	TER : 0.30	HIS : 0.24
ENT : 0.42	EST : 0.28	OFT : 0.22
ION : 0.42	ERS : 0.28	ITH : 0.21
HER : 0.36	ATI : 0.26	FTH : 0.21
FOR : 0.34	HAT : 0.26	STH : 0.21
THA : 0.33	ATE : 0.25	OTH : 0.21
NTH : 0.33	ALL : 0.25	RES : 0.21
INT : 0.32	ETH : 0.24	ONT : 0.20



```
[11/03/22]seed@VM:~/encyption_lecture$ tr 'y' 't' < ciphertext.txt > output.txt
```



Translate ciphertext.txt, and replace all **y** with **t**

```
[11/03/22]seed@VM:~/encyption_lecture$ cat output.txt
tdq uftiqoobxrk lrcqx tqot fo r kwgtfotrbq rqxepfc crlrcftt tqot tdr̄t lxebxqoofvqgt bqto kexq mfuufcwgt ro ft ceitfiwqo. tdq tsqitt kqtqx lrcqx tqot sfgg pqbfi fi tdfxtt
oqceimo. gfiq wl rt tdq otrxt. tdq xwiifib olqqm otrxto ogesgt, pwt bqto urotqx qrcd kfiwtq rutqx tew dqr̄x tdfo ofbirg pqq̄l r ofibgq gr̄l odewgm pq ceklgqtqm qrcd tfkq t
ew dqr̄x tdfo oewim. [mfib] xqkqkpqx te xwi fi r otxrfbdt gfiq, rim xwi ro geib ro leoofpgq. tdq oqceim tfkq tew urfg te ceklgqtq r gr̄l pquexq tdq oewim, tewx tqot fo evq
x. tdq tqot sfgg pqbfi ei tdq sexm otrxt. ei tewx kr̄xj, bqt xqrmt, otrxt.
```

```
[11/03/22]seed@VM:~/encyption_lecture$ tr 'yd' 'th' < ciphertext.txt > output.txt
```

Translate ciphertext.txt, and replace all **y** with **t**, and replace all **d** with **h**

```
thq uftiqoobxrk lrcqx tqot fo r kwgtfotrbq rqxepfc crlrcftt tqot thrt lxebxqoofvqgt bqto kexq mfuufcwgt ro ft ceitfiwqo. thq tsqitt kqtqx lrcqx tqot sfgg pqbfi fi thfxtt
oqceimo. gfiq wl rt thq otrxt. thq xwiifib olqqm otrxto ogesgt, pwt bqto urotqx qrch kfiwtq rutqx tew hqr̄x thfo ofbirg pqq̄l r ofibgq gr̄l ohewgm pq ceklgqtqm qrch tfkq t
ew hqr̄x thfo oewim. [mfib] xqkqkpqx te xwi fi r otxrfbht gfiq, rim xwi ro geib ro leoofpgq. thq oqceim tfkq tew urfg te ceklgqtq r gr̄l pquexq thq oewim, tewx tqot fo evq
x. thq tqot sfgg pqbfi ei thq sexm otrxt. ei tewx kr̄xj, bqt xqrmt, otrxt.
```

Keep adding more characters to your decryption scheme until you get the full answer ☺

# Review the XOR operator:

Everything on a computer is **zeros** and **ones**



0101010101010010111101010100  
1000010111001000101010101100  
1010101010111110100100101010  
1010010101010110010101011010  
1001010101010101010101001010  
101010101010001010101010101  
01011010010101010100101...

Hello world 

01101000 01100101 01101100  
01101100 01101111 00100000  
01110111 01101111 01110010  
01101100 01100100 00001010



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

$1 \oplus 0 = 1$   
 $0 \oplus 0 = 0$   
 $1 \oplus 1 = 0$   
 $0 \oplus 1 = 1$

Message:

$\oplus$  0001 1010 0011  
1100 1100 0101

Key:

Ciphertext:

1101 0110 0110

How to get original message?



# Review the XOR operator:

Everything on a computer is **zeros** and **ones**



0101010101010010111101010100  
1000010111001000101010101100  
1010101010111110100100101010  
1010010101010110010101011010  
1001010101010101010101001010  
101010101010001010101010101  
01011010010101010100101...

Hello world →

01101000 01100101 01101100  
01101100 01101111 00100000  
01110111 01101111 01110010  
01101100 01100100 00001010



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

1 ⊕ 0 = 1  
0 ⊕ 0 = 0  
1 ⊕ 1 = 0  
0 ⊕ 1 = 1

Message:

⊕ 0001 1010 0011  
1100 1100 0101

Key:

Ciphertext:

⊕ 1101 0110 0110  
1100 1100 0101

XOR with the key again!

0001 1010 0011

# Block Cipher

Split in messages into fixed sized blocks, encrypt each block separately

Hello there world

01101000	01100101	01101100
01101100	01101111	00100000
01110100	01101000	01100101
01110010	01100101	00100000
01110111	01101111	01110010
01101100	01100100	00001010

Block 1

Block 2

Block 3

$\oplus$

$\oplus$

$\oplus$



The specifics of this operation vary depending on your mode of encryption

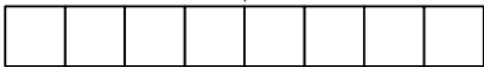
Key

*n bits*

Plaintext



Block Cipher Encryption



Ciphertext

*n bits*

**Decryption** is performed by applying the reverse transformation to ciphertext blocks



- Even small differences in plaintext result in different ciphertexts
- Blocks in plaintext that are the same will also have matching ciphertexts



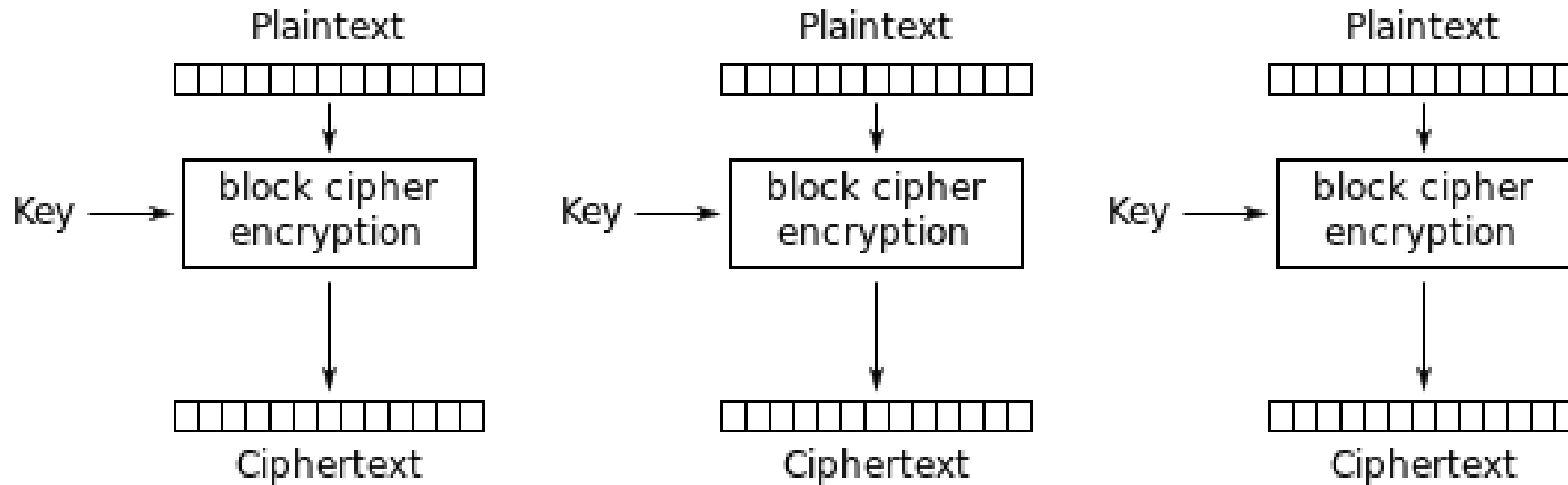
# Modes of Encryption

- Electronic Codebook (ECB)
- Cipher Block Chaining (CBC)
- Propagating CBC (PCBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)

**All block ciphers!**

*But if we aren't careful about how we conduct encryption operations, we may accidentally reveal information about the plaintext*

# Electronic Codebook **ECB**



Electronic Codebook (ECB) mode encryption

**Notice:** For the same key, a plaintext always maps to the same ciphertext

# Using OpenSSL to encrypt w/ ECB

*Encrypt a .txt file*

```
openssl enc -aes-128-ecb -e -in plain.txt -out cipher.txt \
-K 00112233445566778899AABBCCDDEEFF
```

- 1 Encrypt using AES (block cipher) with mode ECB using a 128-bit key
- 2 **Encrypt**
- 3 Input file to be encrypted will be *plain.txt*
- 4 Output file created that contains the ciphertext will be *cipher.txt*
- 5 Key used for encryption will be 00112233445566778899AABBCCDDEEFF 32 characters in hex → 128 bits


# Using OpenSSL to encrypt w/ ECB

*Encrypt a .txt file*

```
openssl enc -aes-128-ecb -e -in plain.txt -out cipher.txt \
-K 00112233445566778899AABBCCDDEEFF
```

*plain.txt*

1 The FitnessGram Pacer Test is a multistage aerobic capacity test that progressively gets more difficult as it continues. The 20 meter pacer test will begin in 30 seconds. Line up at the start. The running speed starts slowly, but gets faster each minute after you hear this signal. [beep] A single lap should be completed each time you hear this sound. [ding] Remember to run in a straight line, and run as long as possible. The second time you fail to complete a lap before the sound, your test is over. The test will begin on the word start. On your mark, get ready, start.



```
[11/09/22] seed@VM:~$ cat cipher.txt
0IeP0%0:00-=600
00=0090z050;NQ0000K0'0po0L?0\2tZ10NQ0i0K000'00mvsJ060L00000*p006n0
0000t0i0Zq000v0p00]00f"0000D0
0000[/0fp0,00p0hyç[000k>
0000000|000>000g)k.0{0+V0;000d000000i
*z%VA;0000lf0v0?00u0$00Z%00T0GfZse
^
0000?C0!00c0JśK0i0Qb00 !C000U0u000>@000)9gm
;00p.~0f0^Ė0?0.0r^00"0000000[000z0;
[0![0 000000aç0_0000E&Di
60yN0?oc00w#0~0000w00?0)+80i03C5:0q00 p800000~/S0Q0[0~5'0+Y0uc0C00
04000aq1Y0000I0000uk00s0000%j070/FP00,x0>0i0X0^0T00zg00C00G000FR,
000fP@|0009h,0{H0g%600@e~0@eZDx'Gp]B/0[11/09/22] seed@VM:~$ █
```

# Using OpenSSL to encrypt w/ ECB

*Encrypt a .txt file*

```
openssl enc -aes-128-ecb -e -in plain.txt -out cipher.txt \
-K 00112233445566778899AABBCCDDEEFF
```

*Decrypt a .txt file*

```
openssl enc -aes-128-ecb -d -in cipher.txt -out new_output.txt \
-K 00112233445566778899AABBCCDDEEFF
```

```
[11/09/22]seed@VM:~$ cat cipher.txt
0IeP0%0:00-=600
00=0090z050;N00000K0'0po0L?0\2tZ10NQ0i0K000'D0mvsJ060L00000*p006n0
0000t0i0Zq000v0p00]00f"0000D0
0000[/0fp0,00p0hyr[000k>
0000?C0!00c0J5K0i0Qb00 !C000U0u000>@000)9gm
;00p.-0f0^E0?0.0r^00"0000000[000z0;
[0![0 00000az0_0000E&Di
60yN0?oc00w#0~0000w00?0)+80i03C5:0q00 p800000^/S0Q0[0~5'0+Y0uc0C00
04000aq1Y0000I0000uk00s0000%j070/FP00,x0>0:0X0^0T00zg00C00G000FR,
000fP@|0009h,0{H0g%600@e~0@eZDx'Gp]B/0[11/09/22]seed@VM:~$ █
```



```
[11/09/22]seed@VM:~$ cat new_output.txt
The FitnessGram Pacer Test is a multistage aerobic capacity test that progressively gets
more difficult as it continues. The 20 meter pacer test will begin in 30 seconds. Line up
at the start. The running speed starts slowly, but gets faster each minute after you hea
r this signal. [beep] A single lap should be completed each time you hear this sound. [di
ng] Remember to run in a straight line, and run as long as possible. The second time you
fail to complete a lap before the sound, your test is over. The test will begin on the wo
rd start. On your mark, get ready, start.
```

# Using OpenSSL to encrypt w/ ECB

*Encrypt a .txt file*

```
openssl enc -aes-128-ecb -e -in plain.txt -out cipher.txt \
-K 00112233445566778899AABBCCDDEEFF
```

*Decrypt a .txt file*

```
openssl enc -aes-128-ecb -d -in cipher.txt -out new_output.txt \
-K 00112233445566778899AABBCCDDEEFF
```

Changing the key used for decryption wont decrypt correctly!

```
[11/09/22]seed@VM:~$ openssl enc -aes-128-ecb -d -in cipher.txt -out new_output.txt -K 00
112233445566778899AABBCCDDEEFF
bad decrypt
140636099929408:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:
crypto/evp/evp_enc.c:583:
[11/09/22]seed@VM:~$ cat new_output.txt
v.00>X!0@.0~hy4c00A}00000(00tg{0M00q00u(00KU00h0%g0zmH0000(000
g'000]0005n00000kD000'L000a00070Vf0(000K0^200J/3;2Y0q00b000&w00-hQ000zY00R+000C0?00j00000
?0'0o00qj?0~A5J/;F.L/D?V00/00f00m00000M00t00H0Dr.#.0
0000i00s*0000&F/000Bv0w=
d0>00r00030i0000r0z
}d00dA00000]F000030000*:0ZX0/0?h0Y0md02W00w05桢0<0z0000r00|0020|0U0bb0[11/09/22]seed@VM:
~$
```