


# Cybersecurity Fundamentals (CS3308/7308)

## Assignment 4 Example Answers

### Question 1 (2point)

1. (2 points) [This file](#)  has been "encrypted" using this simple crypter code below. "Decrypt" the file and find the secret. Is this a good encryption scheme (if the "key" was kept secret)?

```
def mycrypto (filename):  
    f = open(filename,"rb")  
    o = open(filename + ".enc", "w")  
    blob = f.read()  
    i = 0  
    key = 13  
    for b in blob:  
        x = ord(b) ^ (i % key)  
        o.write(chr(x))  
        i += 1
```

The “**^**” **operator** is logical XOR operator in Python (same as in the C language), and the “**%**” **operator** is the modulo operator. So the mycrypto function is taking the first byte of the plaintext, XOR’ing with 0, then the next one with 1, 2, ... 12, then cycle back to 0 (because key is 13). As we have learnt,  $(A \wedge k) \wedge k = A$ , so to “decrypt” the ciphertext, you just need to XOR again with 0,1,...,12 in sequence.

In fact, the mycrypto function can also be used to decrypt an encrypted file. Run the file through the same algorithm and you get back this picture, so the secret is “**eyepiece sniffer overshoe**”.



Is this a good encryption scheme if key (13 in this case) is kept secret? It’s bad for several reasons.

- (1) The key space is small (254 since we are only encrypting byte-wise) so you can just try (i.e., brute-force) all 254 keys until you get the right key. Compare this with, say, AES128 which has  $2^{128}$  key space.
- (2) that’s bad enough on its own. Furthermore, the fact that it’s simple XOR means that it is vulnerable to chosen plaintext attack; the key can be recovered by **ciphertext  $\oplus$  plaintext**. For example, The first 8 bytes of a PNG file is known to be 0x89 50 4E 47. If you XOR this with the first 8 bytes of the ciphertext, you get **0x00 01 02 03 04 05 06 07**, which reveals the pattern of the encryption key.

## Question 2 (2 point)

(2 points) A short, plaintext message has been encrypted using Textbook RSA (using the public exponent) with the following parameters:

```
n=9B51C20306EDE535C8FCAADBC3F3515E52A0D005703DD449BEC66B23E2932313
p=C5A047A7C52ED3A2875F7D76C47B555F
q=C93268355C09197BBF1659B5522FFACD
e=010001
d=0D067636BAC6088AD2281E4BFFCACFEFEF9BC1A69FB9E701063DFBAAB436E4C1
encrypted_message=4A070566DB88A19A8C1212E41DCE3AE42112A8388DA3872EE44AF4C8E654A198
```

This is pretty much just applying the code provided in the workshop, with addition of converting decrypted int into string byte-wise (be careful that just doing `str(65)` converts, for example, 65 to "65" instead of "A" (x41).

```
import binascii
# Copy supplied RSA parameters
p = int("C5A047A7C52ED3A2875F7D76C47B555F",16) # first prime
q = int("C93268355C09197BBF1659B5522FFACD ",16) # second prime
e = int("010001",16) # a number that is co-prime with (p-1)*(q-1)
d = int("0D067636BAC6088AD2281E4BFFCACFEFEF9BC1A69FB9E701063DFBAAB436E4C1")
enc = int("4A070566DB88A19A8C1212E41DCE3AE42112A8388DA3872EE44AF4C8E654A198", 16)
n = p*q
# Print n, d and d*e mod
print "n is: " + str(n)
print "d is: " + str(d)
print "checking d*e mod (p-1)*(q-1): " + str(((d*e) % ((p-1)*(q-1))))

# decrypt message using private exponent d
plain = pow(enc, d, n)
# Convert plain (in int) to str byte-wise by first representing int as hex, then c
onverting that to ascii
print "Cipher " + str(enc) + " is decrypted to: " + binascii.unhexlify("%x" % plain)
n)
```

The output is **Cowanbanga!** (sorry it should have been *Cowabunga* for those Mutant Ninja Turtles fans out there).

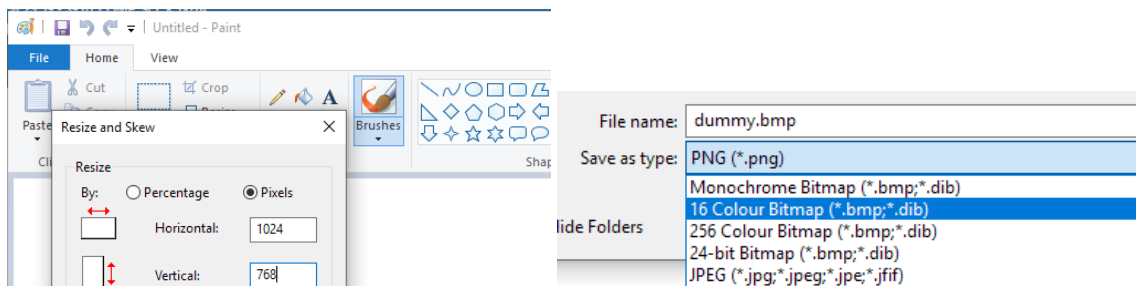
```
n is: 70252945163054194920847511977408156476811805984949774253562479404028773212947
d is: 5891483995546727120328637652438091886928667918827959299068453294388039574721
checking d*e mod (p-1)*(q-1): 1
cipher 33483556006381054149613526142669559072056027658400581727028836768034134991256 is decrypted to: Cowanbanga!
```

## Question 3 (2 point)

Find the message hidden inside this [encrypted image file](#).  Do not try to decrypt it!

This is just a repeat of the cryptography workshop, where we used AES in ECB mode to demonstrate that same blocks of plaintext encrypt to the same ciphertext, thus revealing patterns of the original file.

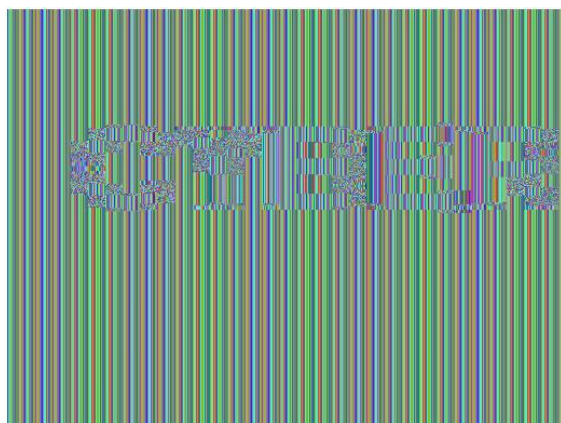
The only difference is that we are not given the original plaintext file (i.e., the image before AES ECB encryption), but we are given a hint in the file name (secret\_16bit\_1024x768). Note: “16-bit” was an error – it should have said “16 color” or “4bit” – sorry if this threw anyone off.



To visualise the encrypted file, we can create (or google as some students have done) a 1024x768 bitmap file and exchange the header portion of the encrypted file with the dummy image.

```
# dd if=dummy.bmp count=54 ibs=1 >> out.bmp
# dd if= secret_16bit_1024x768_ecb.bmp.enc skip=54 ibs=1 >> out.bmp
```

This header copy produces this bmp file below. You can just make out the secret work **cyber**.



## Question 4 (2 point)

(2 points) Decipher this text. What is the cipher, and what is the key?

OM CAL MIT ZTLM GY MODTL, OM CAL MIT CGKLM GY MODTL, OM CAL MIT AUT GY COLRGD, OM CAL MIT AUT GY YGGSOLIFTLL, OM CAL MIT THGEI GY ZTSOTY, OM CAL MIT THGEI GY OF EKTRWSOMB, OM CAL MIT LTALGF GY SOUIM, OM CAL MIT LTALGF GY RAKQFTLL, OM CAL MIT LHKOFU GY IGH, OM CAL MIT COFMTK GY RTLHAOK, CT IAR TXTKBMIOFU ZTYGKT WL, CT IAR FGMIOFU ZTYGKT WL, CT CTKT ASS UGOFU ROKTEM MG ITAXTF, CT CTKT ASS UGOFU ROKTEM MIT GMITK CAB - OF LIGKM, MIT HTKOGK CAL LG YAK SOQT MIT HKTLTLM HTKOGK, MIAM LGDT GY OML FGOLOTLM AWMIGKOMOTL OFLOLMTR GF OML ZTOFU KTETOXTR, YGK UGGR GK YGK TXOS, OF MIT LWHTKSAMOX RTUKTT GY EGDHAKOLGF GFSB.

This seems to be a simple substitution cipher. Well done to students who have diligently applied frequency analysis. A simple analysis can be done using awk as follows, using which a simple text replace can be executed based on known character frequency in English (ETAOIN SHRDLU).

```
$ awk -vFS="" '{for(i=1;i<=NF;i++)w[$i]++}END{for(i in w) printf "%s %d\n", i,w[i]}' ciphertext.txt | sort -r -k 2 -g
T 69
```

M 48  
 O 45  
 G 44  
 L 42  
 I 28  
 A 28  
 K 27  
 F 22  
 C 21  
 Y 19  
 , 17  
 R 14  
 U 13  
 S 12  
 H 10  
 E 7  
 Z 5  
 X 5  
 W 5  
 D 5  
 B 4  
 Q 2  
 . 1  
 - 1  
 119

Unfortunately, this does not yield anything good...

\$ tr [TMOGLIAKFCYRUSHEZXWDBQ] [ETAOINSHRDCUMFPGWYBVKZJQZ] < ciphertext.txt  
**AT DSI TNE WEIT OC TAVEI, AT DSI TNE DOHIT OC TAVEI, AT DSI TNE SME OC DAIUOV, AT DSI  
 TNE SME OC COOFAINREII, AT DSI TNE EPOGN OC WEFAEC, AT DSI TNE EPOGN OC  
 ARGHEUBFATK, AT DSI TNE IESIOR OC FAMNT, AT DSI TNE IESIOR OC USHZREII, AT DSI TNE  
 IPHARM OC NOPE, AT DSI TNE DARTEH OC UEIPSAH, DE NSU EYEHKTNARM WECOHE BI, DE NSU  
 ROTNARM WECOHE BI, DE DEHE SFF MOARM UAHEGT TO NESYER, DE DEHE SFF MOARM  
 UAHEGT TNE OTNEH DSK - AR INOHT, TNE PEHAOU DSI IO CSH FAZE TNE PHEIERT PEHAOU,  
 TNST IOVE OC ATI ROAIAEIT SBTNOHATAEI ARIAITEU OR ATI WEARM HEGEAYEU, COH MOOU  
 OH COH EYAF, AR TNE IBPEHFSTAYE UEMHEE OC GOVPSHAIOR ORFK.**

Fortunately, there are many online services that cracks substitution cipher for you based on character and word frequency. Using <https://www.guballa.de/substitution-solver>, the plaintext is text from Charles Dickens' "A Tale of Two Cities".

## » Substitution Solver «

This tool solves [monoalphabetic substitution ciphers](#), also known as [cryptograms](#). These are ciphers where each letter of the clear text is replaced by a corresponding letter of the cipher alphabet.

As an example here is an English cryptogram this tool can solve:

Rbo rpkrtigo vcrb buucja wj kioj hcjd, km sktpgo, cq rbwr loklgo  
vcgg cjacor kj skhcja wgcja wjd rpycja rk ltr rbcjaq cj cr.  
-- Ropyy Lpursborn

If you want to break a [polyalphabetic cipher](#) instead try the [Vigenère Solver](#).

**Input**

Cipher Text:

ON CAL MIT ZTIR GI ROUTE, ON CAL MIT CORMI GI ROUTE, ON CAL MIT  
AUT GI COLRED, ON CAL MIT AUT GI YGSSOLITILL, ON CAL MIT THOEI GI  
ZTSOT, ON CAL MIT THOEI GI OFERTWOMB, ON CAL MIT LIZALOF GI  
SOUTM, ON CAL MIT LIZALOF GI RAKQITLL, ON CAL MIT LHXOFU GI IGT,   
ON CAL MIT CQNTK GI ROLHACK, CT IAR TKTREMIOPU ZTYGT WL, CT IAR  
FMDIOPU ZTYGT WL, CT CHT ASS UGOFU RORTEM NG ITANPE, CT CHT ASS  
UGOFU RORTEM MIT GMITK CAB - OF LIGRM, MIT HTFOGR CAL LG YAR SOQT  
MIT HRTILFM HTFOGR, MIAM LGUT GI OML FOLOLIM AMNIGHOMOTL OFLOLIMTR  
OF OML ZTGFU KRETKOR, YGR UGR GR YGR ZKOS, OF MIT LHTYBAMONT  
RTUTTT GI ECHMAGLOF GPSS.

Language: English

Break Cipher Clear Cipher Text

**Result**

Clear text [\[hide\]](#)

Key: abcdefghijklmnopqrstuvwxyz This clear text ...  
azertyuiongsdfghjklmxcpbv ... maps to this cipher text

Clear text:

IT WAS THE BEST OF TIMES, IT WAS THE WORST OF TIMES, IT WAS THE  
AGE OF WISDOM, IT WAS THE AGE OF FOOLISHNESS, IT WAS THE EPOCH OF  
BELIEF, IT WAS THE EPOCH OF INCREDULITY, IT WAS THE SEASON OF  
LIGHT, IT WAS THE SEASON OF DARKNESS, IT WAS THE SPRING OF HOPE,  
IT WAS THE WINTER OF DESPAIR, WE HAD EVERYTHING BEFORE US, WE HAD  
NOTHING BEFORE US, WE WERE ALL GOING DIRECT TO HEAVEN, WE WERE ALL  
GOING DIRECT THE OTHER WAY - IN SHORT, THE PERIOD WAS SO FAR LIKE  
THE PRESENT PERIOD, THAT SOME OF ITS NOISIEST AUTHORITIES INSISTED  
ON ITS BEING RECEIVED, FOR GOOD OR FOR EVIL, IN THE SUPERLATIVE  
DEGREE OF COMPARISON ONLY.

The encryption key is

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

The decryption key is the reverse

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

Note that some letters of the alphabet are not represented in the text: J,N,P, and V (corresponding to Q, J, X, Z in plaintext). The substitution is based on the [AZERTY keyboard layout](#) used in French-speaking parts of the world.

## Question 5 (1 point)

Download [this shadow file](#) (/etc/shadow) from an old Linux system. Crack the password for the account called "superman" (the last entry). SSH to **10.0.0.32** (metasploitable.hacklab) using the cracked password to get the the hidden secret file (and its contents).

- Please use hashcat and the rockyou.txt (found under /usr/share/wordlist). You can use other tools if you like.
- The \$1\$ prefix of the hash means it's.... MD5!
- You need to cut the hash string (starting from \$1\$ up to the next ":") and save to a file
- Run hashcat like this (where "-m 500" specifies MD5 as the hashing method)

```
# hashcat -m 500 -a 0 -o <crackedfile> <inputfile> <wordlist> -O --force
```

- How long did it take? (copy the "Started:" and "Stopped:" date and time)

After downloading the shadow file, extract just the password hash with the version and save to a file called hash.

```
$1$2ha70l3z$3R8XkbYAis2F27gzbm4Mr1
```

You may have to unzip the wordlist if you have not done this previously.

```
root@kali:/usr/share/wordlists# gunzip rockyou.txt.gz
```

Run the hashcat as instructed.

```
root@kali:~# hashcat -m 500 -a 0 -o cracked hash /usr/share/wordlists/rockyou.txt --force
hashcat (pull/1273/head) starting...

OpenCL Platform #1: The pocl project
=====
* Device #1: pthread-Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz, 1502/1502 MB allocatable, 1MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Single-Hash
* Single-Salt

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.
Watchdog: Temperature retain trigger disabled.

* Device #1: build_opts '-I /usr/share/hashcat/OpenCL -D VENDOR_ID=64 -D CUDA_ARCH=0 -D
VECT_SIZE=4 -D DEVICE_TYPE=2 -D DGST_R0=0 -D DGST_R1=1 -D DGST_R2=2 -D DGST_R3=3 -D
DGST_ELEM=4 -D KERN_TYPE=500 -D _unroll -cl-std=CL1.2'
Dictionary cache building /usr/share/wordlists/rockyou.txt: 33553434 bytes (23.9Dictionary cache
building /usr/share/wordlists/rockyou.txt: 134213744 bytes (95.Dictionary cache built:
* Filename...: /usr/share/wordlists/rockyou.txt
* Passwords..: 14344392
* Bytes.....: 139921507
* Keyspace...: 14343297
* Runtime....: 2 secs

- Device #1: autotuned kernel-accel to 224
- Device #1: autotuned kernel-loops to 142
[s]tatus [p]ause [r]esume [b]ypass [c]heckpoint [q]uit => [s]tatus [p]ause [r]es
Session.....: hashcat
Status.....: Cracked
```

```
Hash.Type.....: md5crypt, MD5 (Unix), Cisco-IOS $1$ (MD5)
Hash.Target.....: $1$2ha7OI3z$3R8XkbYAis2F27gzbm4Mr1
Time.Started.....: Thu Apr 25 08:36:03 2019 (37 secs)
Time.Estimated....: Thu Apr 25 08:36:40 2019 (0 secs)
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.Dev.#1.....: 2757 H/s (10.08ms)
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 100377/14343297 (0.70%)
Rejected.....: 25/100377 (0.02%)
Restore.Point....: 100153/14343297 (0.70%)
Candidates.#1....: quianna -> oneheart
HWMon.Dev.#1.....: N/A
```

Started: Thu Apr 25 08:36:01 2019

Stopped: Thu Apr 25 08:36:41 2019

root@kali:~# cat cracked

\$1\$2ha7OI3z\$3R8XkbYAis2F27gzbm4Mr1:pretzels

The password for the account "superman" is "pretzels. This took 40 seconds on my VM. SSH into 10.0.0.32 and find the hidden secret.

```
[centos@openvpn-backup ~]$ ssh superman@10.0.0.32
superman@10.0.0.32's password: (type pretzels)
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
```

The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/\*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.

To access official Ubuntu documentation, please visit:

<http://help.ubuntu.com/>

Last login: Thu Apr 11 19:25:54 2019 from 10.0.0.17

superman@metasploitable:~\$ ls -ls

total 0

superman@metasploitable:~\$ ls -a

. .. .aptitude .bash\_history .bash\_logout .bashrc .profile .secret .ssh

superman@metasploitable:~\$ cat .secret

```
/ unseat bankbook unawares suitcase \  
\ reputed poodle /
```

-----

```
\
  /\_)o<
 | 0 . 0|
  \____/
```

## Question 6 (1 point)

Do the same thing with [this shadow file](#) from a more modern Linux system.

- The algorithm is SHA512 (-m 1800) with default 5000 rounds of "stretching" (hash of hash of hash of .... repeated 5000 times) and there is also salt
- The --force switch may not be needed if you are running Kali with GPU support
- Have a coffee or go for a walk or both!
- How long did it take? (copy the "Started:" and "Stopped:" date and time)
- Note that the passwords for Q5 and Q6 are located near the same location in rockyou.txt
- You don't have to login anywhere for this question -- just write the cracked password

Again extract the hash starting with \$6\$ for superman to a file called hash2

```
$6$2Uy.9FAB$KcPA/RKfHt.PT8GuD3sysmV9PBskXH3AI9YWKhuTQRwbxoicZEMt5PSyzHiMxW0Y.vPwVaJkMiD3WS1/IRup.
```

Run hashcat as per instructions, using "-m 1800" for SHA512 with default 5000 stretching.

```
root@kali:~# hashcat -m 1800 -a 0 cracked hash2 /usr/share/wordlists/rockyou.txt --force
hashcat (pull/1273/head) starting...

OpenCL Platform #1: The pocl project
=====
* Device #1: pthread-Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz, 1502/1502 MB allocatable, 1MCU

<SNIP>

Session.....: hashcat
Status.....: Cracked
Hash.Type.....: sha512crypt $6$, SHA512 (Unix)
Hash.Target.....: $6$2Uy.9FAB$KcPA/RKfHt.PT8GuD3sysmV9PBskXH3AI9YWKhu.../IRup.
Time.Started.....: Thu Apr 25 09:58:22 2019 (28 mins, 53 secs)
Time.Estimated....: Thu Apr 25 10:27:15 2019 (0 secs)
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.Dev.#1.....: 58 H/s (7.03ms)
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.....: 100213/14343297 (0.70%)
Rejected.....: 25/100213 (0.02%)
Restore.Point....: 100169/14343297 (0.70%)
Candidates.#1....: pumukli -> popcan
HWMon.Dev.#1.....: N/A

Started: Thu Apr 25 09:58:04 2019
Stopped: Thu Apr 25 10:27:16 2019
root@kali:~# cat cracked
$1$2ha7OI3z$3R8XkbYAis2F27gzbm4Mr1:pretzels
```



\$6\$2Uy.9FAB\$KcPA/RKfHt.PT8GuD3sysmV9PBskXH3AI9YWKhuTQRwbxoicZEMt5PSyzHiMxW0Y.vP  
wVaJjkMiD3WS1/IRup.:**princeoftennis**

It took nearly 30 minutes on my VM... usually a bit faster. The password for the account “superman” was “**princeoftennis**”.