CSc 466/566 Computer Security

Assignment 2

Due Noon, March 8, 2012
Worth 10% (ugrads), 5% (grads)

Christian Collberg
Department of Computer Science, University of Arizona

Introduction

In this assignment we'll examine traditional cipers, electronic mail, and public key cryptography.

1. Part A and B are individual assignments.
2. In part C, you can work in teams of 2 students.

Part A: Traditional ciphers

Your job is to crack the ciphers below. All cleartexts are taken from popular (or not so popular) books, songs, movies, or TV shows. The quotes can all be found on-line. Submit the solution as a list of where the quotes are taken from, like this:

/40

1. Casablanca, the scene at the end where Bogey is telling Ingrid Bergman (the most beautiful woman who ever lived): . . . ;
2. Van Morrison's *Caravan* (AKA, the best song ever written);
3. Sex and the City, episode 45, where Carrie is telling Big: . . . ;
4. Ayn Rand's *Atlas Shrugged*, Chapter 13.

(No, these aren't the actual answers.)
Also include a brief description of how you solved the exercises.
You can use any tools and techniques you want, including

- paper and pencil,
- code you find on-line,
- code you write yourself,

but excluding

- breaking and entering (physically or electronically),
- rubber-hose cryptography,
- help from other humans.

In all cases, the cleartext consists only of the letters A-Z. Blanks and punctuation have been removed.

1. A polyalphabetic substitution cipher where the key-length is 3:

```
KHORFAVHUDFENZECBPYVWVWPFCAUWOGJATMPKAAVVATSOCPWPYPTOJUOY
VWKPKDGFAKBPGFAUHEUQKODQVSZACQMBKYHDQIOCBZUOZCMPJSYQALWHA
TSJFGQRKEVVPJSOGTNCQPKCJUCBCQAPHSJWYJWPWGQCZHAFKWBZUCBHKD
CHPJWOFCAUWOVOGGGPJCOGZEVHHGFAOOEPRATGWPRLWHOVVAOWJVCWPOY
ECQPHPJWOUCQPROHOIKZECFUGODVVAAREFWPKBOWDATAWPHDTSATWCJHQ
PRATFWVSZOCRKSWEHQCZHA
```

2. A monolphabetic substitution cipher:

```
XQVNLXFXPXPTGGTHOXKGBZAMMABLZHTMMHTYZATGBLMTGIEXTLXXQVNLXF
XBMLTGTYZATGBLMTGZHTMLHBMVTGMLMTRAXKXHKXELXBMEEVAHDXHGMAXL
PXXMTBKHYYKXXWHFRXTATYZATGBLMTGBFLHKKRUHRLUNMHNKIETGXLTKXG
MYERBGZMAXKXMAXRKXGHMMAXHGERIETGXLZHBGMHTYZATGBLMTGTKXMAXF
BEBMTKRIETGXLHOXKTMMAXUTLXBFLHKKRPXEERXTAMAXFBEBMTKRIETGXL
VHFXHGZHTMPXEEINMRHNHGHGXHYMAXF
```

3. A homophonic cipher:

```
49 71 06 18 75 04 55 10 23 50 95 14 60 87 35 48 58 45 48 11 23 89 10 08 48
76 11 62 51 24 33 29 95 93 57 02 84 03 53 71 48 84 78 22 53 64 47 17 25 89
96 71 02 42 19 38 07 77 30 36 23 59 10 11 48 46 53 41 16 46 29 59 37 53 91
05 30 56 14 77 61 93 37 16 13 51 03 71 26 48 33 16 19 25 87 20 05 05 29 86
66 05 85 04 70 66 26 93 29 16 69 04 48 95 13 84 01 02 54 20 75 56 08 40 24
22 51 47 58 30 36 07 46 56 55 35 76 57 41 74 84 69 38 53 19 69 48 40 24 11
33 62 30 61 82 35 82 45 70 91 43 56 44 34 32 26 01 34 83 58 79 86 56 44 10
48 27 29 11 67 33 35 72 86 42 95 56 24 37 67 76 81 20 14 46 54 59 59 29 10
56 14 35 08 78 60 90 50 12 57 46 29 90 37 95 73 24 68 47 22 09 26 50 84 68
96 05 59 37 83 80 12 47 20 27 88 91 23 98 29 43 72 12 06 37 55 05 66 32 51
11 23 82 11 22 70 27 89 42 34 62 20 29 19 17 73 89 07 85 73 44 77 34 78 52
62 36 75 23 57 26 48 11 43 42 64 32 71 86 53 70 20 86 53 91 01 75 04 98 37
07 59 58 46 02 01 16 80 62 10 53 52 31 35 59 37 81 27 13 41 29 79 50 48 37
05 51 03 09 13 52
```

4. A polyalphabetic substitution cipher where the key-length is unknown:

```
FFYBVWLGNQJEGIEKJMOVSGVTZUNUPCUJEKIGUKCBVPUVTGEAGUIDGRVFFNULF
DRJVWBSOZEEUFFZZTJPAJVTUFCNFSNUTEZXFDYMVMRJRKFYOFRBOAYFJKBEQU
emacsICVTKJIBOEEULGNCNSWREBHIOMFRWFMFGSLDLZRRTPIFXSIJCVYAADSDDSPZS
ZJVVWSUVTBCIRIMDFT
```

5. Playfair:

```
KX TE WB WF NU NT WC AG LT
SY HZ WG MC TU XK CT ZH EI
EI NF GC AN ED WB QC XY EM
BU UK TY SW IE MH HI NF ZS
ST KX BD AG NI YS YS GX GT
CD WB QC QU MA TG OZ EW NA
FW PM CW ZH EI GR CH GT HZ
SB SW DX NA TN UK MO TG TG
HM WM KA NC GR CH GT HZ SB
HT TU WE XK GT HV MX KW YH
XG DT UP VQ YS FG EQ WY EM
```

```
BU BD PH XN YP XG YD SY HZ
WG HT NF GM CY HY HL CI IE
CT UV MG ZC WB ZW EX YC CN
ZC WB FK MH CT UD AE EI UD
WX BR BV IC AN YS FB YW SL
IK CN WY MW AN XD GM WC NX
AG VB YT UD SP IM GE IM XE
KU OM NH TY HD ZS TM CE MH
HI NF ZS HG NH TY WC PO NE
GT CW IT GL MS VB CE EW XD
TC MK CH DH DX CO TD HZ QV
TU WE MZ NE HC IK KU TU WE
MK ZH GT GE DX CH CT TK CD
WB QC XY EM BU UK TY OZ EX
MX UK EM LH HI NF ZS SH OE
NE AE XE RB MT EX VS MN ZS
SI PE XY XF EG WY EN FB PY
NE LH HI NF ZS IG AE CW PC
YP AG EU
```

NOTE: You can find the ciphertexts on the class website, http://www.cs.arizona.edu/~collberg/Teaching/466-566/2012/Assignments/index.html.

Part B: Encrypted email

Here we'll play with sending and receiving encrypted email. /10

1. Install pgp on your machine.
2. Install thunderbird (http://www.mozilla.org/en-US/thunderbird/) on your machine.
3. Set things up so that you can send/recieve encrypted emails.
4. Get the TA's public key from D2L.
5. Send the TA (nitinshinde@email.arizona.edu) a message (encrypted with his public key) containing your own public key.
6. The TA will respond with a message containing a random number R encrypted with your public key.
7. Respond to him, with an encrypted message containing the number $R + 1$.

Part C: Public-Key Encryption

Implement the RSA algorithm to generate key pairs, encrypt a file, and decrypt a file. Implement your program in Java. The mathematical operations related to the algorithm implementation can be done using the java.math.BigInteger class. /50

Your program should support the following operations:

1. java RSA -h
   - This should list out all the command line options supported by your program.
2. java RSA -k <key_file> -b <bit_size>:

- This should generate two text files <key_file>.public and <key_file>.private containing the public and private keys, respectively, encoded in hex.
- The size of the key should be <bit_size> bits. You should support at least the sizes 512, 1024, and 2048 bits.
- In case -b option is not specified, the default bit size should be 1024.
- Each time this mode is executed, different key pairs must be generated, i.e., you must extract some entropy from the environment.

3. java RSA -e <key_file>.public -i <input_file> -o <output_file>:

    - This should encrypt the file <input_file> using <key_file>.public and store the encrypted file in <output_file>.
    - There is no restriction on the size if the input file.

4. java RSA -d <key_file>.private -i <input_file> -o <output_file>:

    - This should decrypt the file <input_file> using <key_file>.private and store the plain text file in <output_file>.

NOTE: Typcially, RSA is used as a part of a hybrid protocol, where it is used to encrypt a symmetric key. Here, for simplicity, we use RSA to encrypt an entire file, which normally would be too inefficient. Note that, because of this, you are expected to break up the input file into appropriate sized blocks, apply appropriate padding, and encrypt each block in ECB mode.

NOTE: You cannot use the java.security package, nor any other code not written by yourselves, to implement this program.

NOTE: In this assignment we rely on the honor code: You cannot look at any RSA implementations. I'm sure there must be billions of implementations on the web (in C, perl, Visual Basic, etc.) and in textbooks, but you cannot look at any of them. You may, of course, read about the algorithm itself — you just can't look at any code.