Homework Assignment # 1

Miguel Gomez U1318856

2024-02-04 23:13:21

Contents

1 Homework 1

The extended euclidean algorithm is a spin on the usual algo that allows for us to split the number into two intermediate values. one that is equal to the number we are dividing in the algo multiplied by a number, and another that is the divisor multiplied by the quotient. This expression below is the one we end up with:

1.1 expression

$$g = gcd(a, b)$$
$$\exists\ s, t \mid s \cdot a + t \cdot b = g$$

```
start=$(date +%s.%N)
Singular sing/hw1_b.sing | grep -v -e "\*\* loaded\|\*\* library"
end=$(date +%s.%N)
echo "Execution Time: $(echo "$end - $start" | bc) seconds"
```

1.2 output of hw1$_b$ results

```
                     SINGULAR                                 /  Development
 A Computer Algebra System for Polynomial Computations       /   version 4.2.1
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   May 2021
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \  Debian 1:4.2.1-p3+ds-1
// ** but for functionality you may wish to change to the new
// ** format. Please refer to the manual for further information.
The example computed GCD of 24 and 16 is:
8
// ** redefining r (ring r = integer, (x), lp;) hw1_b.sing:21
The computed myintGCD of 24 is: 8
The computed myEuclid of 24 is: 8
The computed myExtendedEuclid of the numbers is:

GCD(24,16) = 8
s = 1
t = -1

The computed GCD of the list of numbers for problem 1-b is:
10
Auf Wiedersehen.
Execution Time: .028753389 seconds
```

1.3 Pseudocode for the Euclidean algo

Algorithm 1 Euclidean Algorithm

procedure MYEXTENDEDEUCLID(a, b)
    $R1 \leftarrow a$
    $R2 \leftarrow b$ **while** $R2 \neq 0$ **do**
      $Q \leftarrow (R1/R2)$
    $r \leftarrow R1 - Q \times R2$
    $R1 \leftarrow R2$
    $R2 \leftarrow r$

    return r
end procedure

1.4 Pseudocode for the Euclidean algo

Algorithm 2 Extended Euclidean Algorithm

```
procedure myExtendedEuclid(a, b)
    R1 ← a
    R2 ← b
    S1 ← 1
    S2 ← 0
    T1 ← 0
    T2 ← 1 while R2 > 0 do

     Q ← floor(R1/R2)
    r ← R1 − Q × R2
    R1 ← R2
    R2 ← r
    s ← S1 − Q × S2
    S1 ← S2
    S2 ← s
    t ← T1 − Q × T2
    T1 ← T2
    T2 ← t

    print "GCD(", a, ",", b, ") = ", S1 × a + T1 × b
    print "s = ", S1
    print "t = ", T1
    L ← list()
    L ← list(S1 × a + T1 × b, S1, T1)
    return L
end procedure
```

1.5 Identify whether the integers 38 and 7 have multiplicative inverses in $\mathbf{Z_{180}}$

Since the number p we are working with is even, we will not have multiplicative inverses for even numbers. Therefore we only need to find the inverse for the one we can, for 7.

$$a \in \mathcal{Z}_{180}\ ,\ a^{-1} \in \mathcal{Z}_{180} \text{ if } gcd(a, 180) = 1$$

```
start=$(date +%s.%N)
Singular sing/hw1_c.sing | grep -v -e "\*\* loaded\|\*\* library"
end=$(date +%s.%N)
echo "Execution Time: $(echo "$end - $start" | bc) seconds"
```

1.6 output of hw1_c results

```
                      SINGULAR                                 /  Development
 A Computer Algebra System for Polynomial Computations       /   version 4.2.1
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   May 2021
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \  Debian 1:4.2.1-p3+ds-1
// ** but for functionality you may wish to change to the new
// ** format. Please refer to the manual for further information.
The computed myintGCD of 7 is:
1
The computed myintGCD of 38 is:
2

GCD(7,180) = 1
s = -77
t = 3

The inverse of 7 modulo 180 is 103

GCD(38,180) = 2
s = 19
t = -4

38 has no inverse modulo 180
Auf Wiedersehen.
Execution Time: .022770642 seconds
```

Since the gcd for the expression comes out to 1, the inverse exists and is printed out. Since 38 is even, no inverse is possible modulo 180.

2 Problem 2

Solving linear diophantine equations using linear congruences.

2.1 a) solving LC $4x \equiv 4$ mod 6

Solving this is easiest with a table of the results we would get by plugging in any values for x from the set mod 6.

x	$4x \mod 6$	Congruent to 4?
0	$4 \cdot 0 \mod 6 = 0$	No
1	$4 \cdot 1 \mod 6 = 4$	Yes
2	$4 \cdot 2 \mod 6 = 2$	No
3	$4 \cdot 3 \mod 6 = 0$	No
4	$4 \cdot 4 \mod 6 = 4$	Yes
5	$4 \cdot 5 \mod 6 = 2$	No

$\therefore$ we have exactly two solutions which are congruent for this problem.

2.2 Solving as an LDE instead

using the expression $4x \equiv 4$ mod 6, we can transform the expression into the following:

$$\begin{aligned}4x &\equiv 4 \text{ mod } 6\\ 6 &| \; 4x - 4\\ 6k &= 4(x-1)\\ 3k &= 2(x-1)\end{aligned}$$

Now we find values of x that would allow the expression to be integer valued when $x \in \{0..5\}$. In general, the solutions will be the same as they were before giving us just two possible solutions to the expression. Using the following:

$$\begin{aligned}x &= 1\\ 3k &= 2(1-1) = 0\\ k &= 0\\ x &= 4\\ 3k &= 2(4-1) = 6\\ k &= 2\end{aligned}$$

3 Problem 3 - Affine Cipher

We must find a solution to the affine cipher and obtain the keys $[k_1, k_2]$. We can set this up in singular to solve for the key vector using the inverse matrix algorithm that utilizes the transformation to a reduced row eschelon form of the matrix.

```
start=$(date +%s.%N)
Singular sing/hw3.sing | grep -v -e \
    "\*\* loaded\|\*\* library\|\*\* redefining"
end=$(date +%s.%N)
echo "Execution Time: $(echo "$end - $start" | bc) seconds"
```

3.1 output of hw3 results

```
                     SINGULAR                                 /  Development
 A Computer Algebra System for Polynomial Computations       /   version 4.2.1
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   May 2021
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \  Debian 1:4.2.1-p3+ds-1

print matrix A
18,1,
19,1

print matrix B
4,
19

Determinant of A:
printing det(A)
25

gcd(det(A), 26) is:
1

inverse of A exists
inverse of A:
25,1,
19,8
check of inv_A*A = I:
1,0,
0,1

Solutions for x = :
15,
20

K[1,1]=15
K[2,1]=20

Mapping given the keys is:
A -> U
B -> J
C -> Y
D -> N
E -> C
F -> R
```

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

Mapping back to plain text uses the reverse list
U -> A
J -> B
Y -> C
N -> D
C -> E
R -> F
G -> G
V -> H
K -> I
Z -> J
O -> K
D -> L
S -> M
H -> N
W -> O
L -> P
A -> Q
P -> R
E -> S
T -> T
I -> U
X -> V
M -> W
B -> X
Q -> Y
F -> Z

The place where bob should meet on vacation is:
NOVASCOTIA
Canada should be nice in a few months.
Auf Wiedersehen.
Execution Time: .060540221 seconds
```

Shown above is the results and we see that the key $[k_1, k_2]$ is $[15, 20]$ and Novascotia is the destination for the vacation.

4 Problem 4 - Hill Cypher

Considering the Hill Cypher, we are restricted to using only 8 letters. Therefore we must do everything modulo 8. Getting the encryption $\mathbf{C} = \mathbf{P} \cdot \mathbf{K}$, and the decryption $\mathbf{P} = \mathbf{C} \cdot \mathbf{K}^{-1}$ will be done as follows:

$$\mathbf{K} = \begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}$$

$$\mathbf{P} = \begin{pmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{pmatrix}$$

To find our inverses, we must do the calculations modulo 8, which means we should employ the gcd method. If we can show the gcd of the determinant and the modulus is 1, then we can invert the matrix. First we tackle part a.

a) Set up the problem as a system of linear congruences to identify $\mathbf{K}$.

$$\begin{pmatrix} 2 \cdot k_{11} + 3 \cdot k_{21} & 2 \cdot k_{12} + 3 \cdot k_{22} \\ 2 \cdot k_{11} + 5 \cdot k_{21} & 2 \cdot k_{12} + 5 \cdot k_{22} \end{pmatrix} = \begin{pmatrix} 4 & 5 \\ 0 & 7 \end{pmatrix}$$

$$\begin{aligned} 2 \cdot k_{11} + 3 \cdot k_{21} &= 4 \bmod 8 \\ 2 \cdot k_{12} + 3 \cdot k_{22} &= 5 \bmod 8 \\ 2 \cdot k_{11} + 5 \cdot k_{21} &= 0 \bmod 8 \\ 2 \cdot k_{12} + 5 \cdot k_{22} &= 7 \bmod 8 \end{aligned}$$

b) Is the given matrix $\mathbf{P}$ invertible? Is the given matrix $\mathbf{C}$ invertible? In other words, can we We apply the encryption algorithm to the plaintext, character by character:
The given matrix P is:

$$\mathbf{P} = \begin{pmatrix} 2 & 3 \\ 2 & 5 \end{pmatrix}$$

The given matrix C is:

$$\mathbf{C} = \begin{pmatrix} 4 & 5 \\ 0 & 7 \end{pmatrix}$$

Are the given matrices invertible? Can we compute the key as

$$\mathbf{C} \cdot \mathbf{P}^{-1} = \mathbf{K} \text{ ?}$$

4.1 a)

```
start=$(date +%s.%N)
Singular sing/hw4.sing | grep -v -e \
    "\*\* loaded\|\*\* library\|\*\* redefining"
end=$(date +%s.%N)
echo "Execution Time: $(echo "$end - $start" | bc) seconds"
```

4.1.1 output of hw4 results

```
                     SINGULAR                                 /  Development
 A Computer Algebra System for Polynomial Computations       /   version 4.2.1
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   May 2021
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \  Debian 1:4.2.1-p3+ds-1
Setting cores:
12
P:
2,3,
2,5
C
4,5,
0,7
expr1
2*k11+3*k21,2*k12+3*k22,
2*k11+5*k21,2*k12+5*k22
expr2
2*k11+3*k21+4,2*k12+3*k22+5,
2*k11+5*k21,  2*k12+5*k22+7
Determinant of P:
printing det(P)
4
Determinant of C:
printing det(C)
4
since the determinant of P and C are both outside of the star-set Z_{8^*}, they are not invertible.
Exiting b/c gcd failed
gcd(det(P), 8): 4
inverse(P)
// ** matrix is not invertible
_[1,1]=0
inverse(C)
// ** matrix is not invertible
_[1,1]=0
Auf Wiedersehen.
Execution Time: .046434783 seconds
```

Given the results here, the given matrices are not invertible.

c) Does there exist a unique (one and only one) key matrix **K** that satisfies these constraints? If not, how many distinct matrices **K** can be used for this cipher?
From my analysis in Singular and in Python, it seems there are 9 possible matrices mod 8 that would allow P to be encrypted as C. So there does not exist only one matrix which could do this.

```
start=$(date +%s.%N)
Singular sing/hw4_method2.sing | grep -v -e \
    "\*\* loaded\|\*\* library\|\*\* redefining"
end=$(date +%s.%N)
echo "Execution Time: $(echo "$end - $start" | bc) seconds"
```

4.1.2 output of hw4$_{method2}$.sing results

```
                     SINGULAR                                 /  Development
 A Computer Algebra System for Polynomial Computations       /   version 4.2.1
                                                           0<
 by: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann     \   May 2021
FB Mathematik der Universitaet, D-67653 Kaiserslautern        \  Debian 1:4.2.1-p3+ds-1
// ** but for functionality you may wish to change to the new
// ** format. Please refer to the manual for further information.
Setting cores:
12
G[1]=k22+7
G[2]=k21+6
G[3]=2*k12+3*k22+3
G[4]=2*k11+3*k21+4
For K to be invertible, the gcd of the determinant mod 8 and 8 should be 1.
Since 8 is even, we expect only odd elements to have an inverse mod 8.
gcd(1,8)= 1
gcd(3,8)= 1
gcd(5,8)= 1
gcd(7,8)= 1
We see that only odd elements have inveerses. Given our K, we can proceed if det(K) is in
     the set Z_8^*.
1,1,
6,1
det(K):3
K matrix found by groebner gives determinant 3 which is in the star-set.
The result of P*K:
4,5,
0,7
Which is equal to the  we expect to get:
4,5,
0,7
Auf Wiedersehen.
Execution Time: .044736744 seconds
```

```
start=$(date +%s.%N)
python3 py/hw4_exhaustive.py
end=$(date +%s.%N)
echo "Execution Time: $(echo "$end - $start" | bc) seconds"
```

4.1.3 output of hw$4_{method2}$.sing results

```
K matrix:
[1 3]
[6 5]
det(K): 3
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
K matrix:
[3 1]
[2 1]
det(K): 1
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
K matrix:
[3 3]
[2 5]
det(K): 1
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
K matrix:
[3 7]
[2 5]
det(K): 1
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
K matrix:
[5 1]
[6 1]
det(K): 7
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
K matrix:
[5 5]
[6 1]
det(K): 7
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
K matrix:
[5 7]
[6 5]
det(K): 7
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
K matrix:
[7 3]
[2 5]
det(K): 5
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
K matrix:
[7 7]
[2 5]
det(K): 5
gcd_mod_det: 1
The result of P*K which is equal to the C we expect to get:
[4 5]
[0 7]
The total number of possible matrices is: 4096
```

```
the final invertible count is          : 1992
the # of final C being correct is      : 9
number possible should be 8^4: 4096
Therefore, the total number of invertible matrices that can
produce C is: 9
percentage of solutions vs total number: 0.220 %
Execution Time: .285150142 seconds
```

d) Based on the above analysis, explain whether the above system is secure to a known-plaintext or a chosen-plaintext attack? [Note: A known-plaintext attack is one where some $(\mathbf{P}, \mathbf{C})$ pairs are known to Eve. A chosen-plaintext attack is similar to the known-plaintext one, except that the $(\mathbf{P}, \mathbf{C})$ pairs are chosen by the attacker herself.]
   Given how quickly were able to find the results here in Python or Singular, I would think this method is not very safe against a plain text attack. Here I was able to locate a set of keys that are invertible, which convert $\mathbf{C} = \mathbf{P} \cdot \mathbf{K}$. One could then find some invertible matrix P and feed that to the encryptor to get C. That C could be multiplied by P^{-1} to get K back.

```
start=$(date +%s.%N)
python3 py/hw4_plaintext_inv.py
end=$(date +%s.%N)
echo "Execution Time: $(echo "$end - $start" | bc) seconds"
```

4.1.4 output of hw$4_{method2}$.sing results

```
P:
[[2 3]
 [2 5]]
K:
[[3 1]
 [2 1]]
P_att_inv:
[[4. 7.]
 [1. 2.]]
C:
[[4 5]
 [0 7]]
C_attac:
[[0 3]
 [5 3]]
P^-1*C = P^-1*P*K = K:
[[6.5  1.5 ]
 [4.75 5.75]]
Execution Time: .226528343 seconds
```

Given this result, it would seem that it may be more difficult to obtain something that works when finding the inverse modulo 8. I would expect that this is faily close, but I am still missing something in the python methods that prevent the solution from working out exactly the correct process for attacking this and showing the recalculation of the keys.