Group 1

AREA A:

Substitution

To encrypt: Create a random alphabet of letters and rearrange plaintext according to those letters. Must be a one-to-one relationship.
To decrypt: Use frequency analysis to find a pattern and try to make words based off the frequency analysis.

Example (*important* must be one-to-one:
A → D
B → B
C → A
D → C

AABD would encrypt to DDBC

Affine Cipher

In order to encrypt using the Affine cipher, you need two values, a and b.
A must be 1,3,5,7,9,11,15,17,19,21,23, or 25 while B can be any positive integer between 0-26
Once you have these two values, you can encrypt your message using the formula

E(x) = $ax + b \; mod \; m$, where m is the size of the alphabet (usually 26) and a & b are values that are defined in the paragraph above.
The decryption algorithm is
D(x) = $a^{-1}(x - b) \; mod \; m$, where m is the size of the alphabet (usually 26) and a & b are values that are defined in the paragraph above.

Encryption: The plaintext ‘KNIGHTS’ translates to the ciphertext ‘WPSODBM’ when using a A value of 15 and a B value of 2.
Decryption: Use the decryption algorithm and find the mod inverse in order to decrypt Affine.

Shift Cipher

(AKA Caesar shift cipher)
Note: For any shift cipher, it is important to know the size of your alphabet and the size of the shift of each letter. Most of the time the size is 26, but it could be 36 if the numbers ‘0’ - ‘9’ are included.

Example:

Let us use an alphanumeric alphabet of the letters ‘A’ through ‘Z’ and the numbers ‘0’ - ‘9’

Write out this alphabet somewhere to make encryption & decryption simple.
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
For encryption: Shift n spaces to the LEFT, where n = the number of spaces to shift
For decryption: Shift n spaces to the RIGHT, where n = the number of spaces to shift

Example: Encrypt ‘CIS3362’ with a shift size of 12.
In this case, the encrypted ciphertext is ‘OU4FFIE’
Decryption Example: Decrypt the ciphertext ‘5YFYAZ2’ with a shift size of 12
The Plaintext is the phrase ‘HARAMBE’

Alternatively, the shift cipher can be encrypted and decrypted using the formula
$E(x) = (x+n) \bmod s$, where x is the letter to be encrypted, n is the shift size and s is the alphabet size (usually 26)
$D(x) = (x-n) \bmod s$, where x is the letter to be decrypted n is the shift size and s is the alphabet size (usually 26)

Euclidean Algorithm

Recall: GCD(A,B) = largest integer that divides both A and B

- If A = 0,

then GCD(A,B) = B
since GCD(A,0) = A

- If B = 0,

then GCD(A,B) = B
since GCD(0,B) = B

- Write A in quotient remainder form (A = B*Q + R)
    - Find GCD(B,R) using the Euclidean Algorithm since GCD(A,B) = GCD(B,R)

Example: (refer back to Week 01 – class notes)
Given: n = 26 , a = 7
26 = 7(3) + 5 gcd(26,7) = gcd(7,5)
7 = 5(1) + 2 gcd(7,5) = gcd(5,2)
5 = 2(2) + 1 gcd(5,2) = gcd(2,1)
2 = 1(2) + 0

Extended Euclidean Algorithm

Take the last step of the algorithm to substitute.
1.) Rearrange each of the equations to equal to the remainders.
2.) Using those remainders, that is what you will substitute in.
Example: (using previous example.. continuation)

Step 1:
5 = 2(2) + 1 à 1 = 5 – 2(2)
7 = 5(1) + 2 à 2 = 7 – 5(1)
26 = 7(3) + 5 à 5 = 26 – 7(3)

Step 2:
1 = 5 – 2(2)
1 = 5 – (7 – 5(1))(2)
= 5 – 2(7)+ 5(2)
= 5(3) – 2(7)
1 = (26 – 7(3))(3) – 2(7)
= 26(3) – 7(9) – 2(7)
1 = 26(3) – 7(11) mod 26 <-- remember that we are mod 26 because n is 26
1 mod 26 = -11(7) <-- recall that -11 mod inverse is 15
15*7 = 1 mod 26
Inverse 7 = 15 mod 26

AREA B:

Vigenere

Encryption:

1. Choose a keyword
2. Determine numerical values of letters in keyword and plaintext (A - Z = 0 - 25)
3. Line up plaintext and keyword, repeating the keyword until it matches the length of the plaintext
4. Add numerical values to retrieve values of ciphertext
5. Mod 26 to convert to values in the range 0 - 25
6. Convert numerical values to alphabetical to get ciphertext

Example (Fall 2016, Exam 1, Question 5):
Encrypt the plaintext “WEAREWORKING” using the Vigenere cipher and the keyword “HOUSE”

	W	E	A	R	E	W	O	R	K	I	N	G	(plaintext)
	22	4	0	17	4	22	14	17	10	8	13	6	(plaintext values)
	H	O	U	S	E	H	O	U	S	E	H	O	(keyword - repeated)
+	7	14	20	18	4	7	14	20	18	4	7	14	(keyword values)
	29	18	20	35	8	29	28	37	28	12	20	20	**mod 26**
	3	18	20	9	8	3	2	11	2	12	20	20	(ciphertext values)
	D	S	U	J	I	D	C	L	C	M	U	U	(ciphertext)

ANSWER:
DSUJIDCLCMUU

Decryption:

Decryption is same process as encryption, except you subtract the values of the keyword from the values of the ciphertext to get the plaintext.
So to get the plaintext value of the first letter in the example above:
'D' = 3, subtract 'H' = 7, (3 - 7 = - 4), mod 26 gives 22, which translates to 'W'

Index of Coincidence

Index of coincidence is used to find the probability of randomly selecting two of the same items from a list.

$$\frac{\sum_{i=0}^{25} f_i(f_i - 1)}{n(n-1)}$$

Where f is the frequency of each item in the list and n is the number of items in the list. The IC of most english text should be at or around .0618, and lower for any random letter substitution.

EXAMPLE

Determine the index of coincidence for the following set of letters:
10 As, 25 Bs, 25 Cs, 40 Ds.

$$IC = \frac{(10*9)+(25*24)+(25*24)+(40*39)}{100*99}$$

$$IC = \frac{90+600+600+1560}{9900}$$

$$IC = \frac{1}{110} + \frac{2}{33} + \frac{2}{33} + \frac{26}{165} = \frac{1}{110} + \frac{46}{165} = \frac{96}{330} = \frac{19}{66}$$

Mututal Index of Coincidence

Given two sets of item, what is the probability that an item chosen from the first set will equal an item chosen from the second second set.

$$\frac{\sum_{i=0}^{25} f_i * g_i}{m * n}$$

Where f is the frequency of the item in the first set, g is the frequency of the item in the second set, m is the total number of items in the first set, and n is the total number of items in the second set.

EXAMPLE

Set A = 10As, 20Bs, 15Cs, 25Ds
SetB = 5As. 25Bs, 10Cs, 10Ds

$$MIC(A,B) = \frac{(10*5)+(20*25)+(15*10)+(25*10)}{70*50}$$

$$MIC(A,B) = \frac{50+500+150+250}{3500} = \frac{1+10+3+5}{70} = \frac{19}{70}$$

PLAYFAIR CIPHER Group 11

(Note: ADFGVX, Transposition missing)

- Created in 1854 by Charles Wheatsstone

Steps

1. pick a keyword that doesn't have repeating letters in it.

   Bad Ex. "HARAMBE" would not work

   Good Ex. "MONEY" would work

2. Create a table with measurements of "5 × 5"

   The keyword would go in first into the table followed by the rest of ~~table~~ alphabet without repeating the letters from the keyword AND excluding the letter "J"

Ex.

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

↓

WRONG TABLE

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

3. Prepare Your Message

Message "I NEED MORE MONEY"

- Split the message into pairs of letters
- seperate duplicate letters by adding an "x" between them
- if there is an odd letter at the end add an "x" to make it even
- Ignore all spaces

Ex. "IN EX ED MO RE MO NE YX"

4. ENCRYPTING RULES

- If ~~the~~ the pair of letters are in the same column move each letter "down one", when you reach end of table wrap it around.
- If pair is in same row, move each letter "Right one", when you reach end of table wrap it.
- If pairs form a "rectangle", swap letters at opposite ends of rectangle.

Ex.

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

Broken up Message

"IN EX ED MO NE YX"

IN	→	C R	"Colm. shift"
EX	→	D E	"Colm. Shift"
ED	→	D K	"Colm. shift"
MO	→	O N	"Row shift"
NE	→	E Y	"Row shift"
YX	→	E Z	"Rectangle switch"

Cipher Text

"CRDEDKONEYEZ"

DECRYPTION

- Must be given a cipher text with the keyword.

- Flip all the encryption rules.

  Colm → Shift ~~down~~ Up

  Row → Shift Left

  Rectangle → doesn't need to flip, switch ends of rectangle

The Enigma machine was used by Nazi Germany to send enciphered messages during World War II. It contained a series of rotors (usually three out of a possible five to choose from) and when one presses a letter on the keyboard, the stepping mechanism would move the rightmost rotor by one twenty-sixth of a full rotation. When that rotor does a full rotation, it kicks the next rotor one place and the right rotor keeps going. Eventually the middle rotor (if there's three rotors) does a full turn and it will kick the left rotor one place. The idea is similar to the hands on a clock. Imagine an hour hand, a minute hand, and a second hand. So then, there's a fast rotor, a middle rotor, and a slow rotor.

To encrypt a message, when one presses a key, the enciphered letter that corresponds to it lights up. Inside the Enigma, the key press steps the rotors and the wires located conveniently inside the rotors connect the battery to a different bulb.

For decrypting some ciphertext, one would need to know the initial starting position of the rotors when the message was encrypted, this concept is similar to an initialization vector in modern cryptography systems. The initial rotor position was usually transmitted just before the ciphertext, usually after having been enciphered. Finger wheels on each rotor protruded through the lid, allowing the operator to tune the rotors to the exact position.

Adding an extra level of scrambling, at the front of the Enigma is a plugboard. There were ten wires, each connecting two letters into a pair. So if Q was plugged into E, those two letters would swap over and when an operator presses E, the signal is diverted to Q before entering the rotors to be scrambled.

One of the conceptual flaws in the Enigma was the reversal rotor, called the reflector, ensured that encryption was the same as decryption. It also meant that no letter ever encrypted to itself and this was subsequently exploited by codebreakers.

Working on the combinations of settings:

Rotors: 5 x 4 x 3 = 60 ways we can put in 3 rotors from a choice of 5

Starting positions: 26 x 26 x 26 = 17576

Plugboard: 26! = number of ways to arrange the 26 letters of the alphabet

We don't want the combination of 26 letters, we only want to make 10 pairs. That leaves 6 letters left over which we don't care about. Since we don't care about them, we can divide by 6!
There are 10 pairs, since we don't care about the order, we can divide by 10!
From the 10 pairs, each pair is one distinct combination. Two pairs, AB and BA are the same pair.
Hence:
Plugboard = 26! / (6! x 10! x 2^10)

Total number of ways to set the Enigma = plugboard x starting positions x rotor combinations
= nearly 159 quintillion settings

Hill Cipher

The Hill Cipher uses matrix multiplication to substitute sets of letters in a message with a key being an NxN matrix.
If there is a 2x2 key such as
K=(3, 6)
(4, 7)
And the plain text is PAPERS. The plain text will be split into pairs for multiplication with the key matrix. If the plain text is PAPER a padding letter should be added to make complete pairs PAPERX
When doing the matrix multiplication each answer should be modulo 26 to get a number that can be associated to the ciphertext letter.
Using the key K and the plain text PAPERS the ciphertext is TIRKDM found by
(3, 6) (P) = (3, 6) (15) = (3 * 15 + 6 * 0) = 45 % 26 = 19 = T
(4, 7) (A) = (4, 7) (0) = (4 * 15 + 7 * 0) = 60 % 26 = 8 = I

This is repeated for every pair of letters till all plaintext is encrypted.

To decrypt we need K inverse
To find this we can use the formula [(ad - bc) % 26] [d, -b]
[-c, a]
So to find the inverse of K
(3 * 7 - 6 * 4) % 26 = -3 % 26 = 17

17 [7, -6] = [119, -102] = [15, 2]
[-4, 3] = [-68, 51] %26 = [10, 25]

To decrypt use K inverse as K
[15, 2] [19] mod 26 = P
[10, 25] [8] mod 26 = A

With the english alphabet Hill cipher has a security of 26 raised to the 4th.

A test example is : there is a Key
K =(6, 5)
(7, 11) with a ciphertext “MEQJ” What is the corresponding plaintext?

To find the plaintext we will first need the inverse key.
(6 * 11 - 5 * 7) = 31 mod 26 = 5

5 [11 -5] = [55 , -25] mod 26 = [3 , 1]
[-7, 6] = [-35, 30] mod 26 = [17 , 4] = inverse K
M = 12 E = 4 Q = 16 J = 9

(3, 1) (12) = (3 * 12 + 1 * 4) = 40 % 26 = 14 = O
(17, 4) (4) = (17 * 12 + 4 * 4) = 220 % 26 = 12 = M
(3, 1) (16) = (3 * 16 + 1 * 9) = 57 % 26 = 5 = F
(17, 4) (9) = (17 * 16 + 4 * 9) = 308 % 26 = 22 = W

So the plaintext is “OMFW”

DES

The 32-bit input is first put through the expansion box to become 48 bit. This new value is XORed with the key, k_i. The resulting value is put through the first s-box and becomes a 32-bit value again. This new value is then put through the permutation matrix, P.

How to do S-box problems

Convert hex to binary
Divide into 6 bits blocks
First and last bit of block are used to determine row of s-box
Middle four bits are used to determine column of s-box
Convert number to hex

How to do E-box problems

Convert the hex input into binary
Following the order in the E-box fill in each binary digit (there will be duplicated bits)
Once all 64 bits of output have been generated, convert back into hex

How to do permutation-box problems

Convert the hex input into binary
Reorder the binary digits according to the initial permutation (ip) table
Convert the binary output back into hex

Sample problems from old exams

For the problems without answers, most answers can be found in past exam solution manuals. Problems are listed so that you will know what type of DES questions the professor normally asks so that you can make sure you know how to do all problem types.

2) (4 pts) If the input in DES to S-box 5 is 001111, what is the output?

01 - 1
0111 - 7
Row 1, Column 7
0001 hex
1 decimal

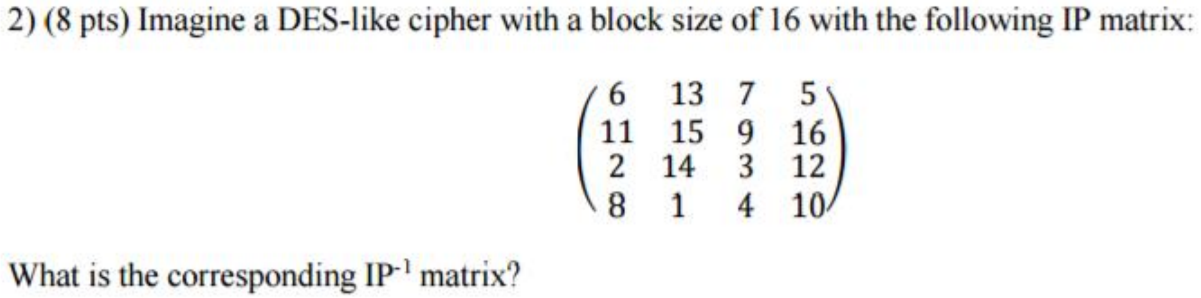
3) (8 pts) The first part of the function F in a round of DES expands the 32-bit input (from the right half of the previous round) to 48 bits. If this input, in HEX to the function F is 7DA839B2, what are the last 8 bits of output right after this value is processed by the Expansion Permutation E?

4) (10 pts) Without examining all entries in the 16 round key schedule of DES, determine whether or not each number (which represents a bit location in the original key in each of the 16 boxes labeled "Round 1" through "Round 16") appears the exact same number of times collectively in the 16 boxes. (As an example, 10 appears in round except rounds 4, 12 and 14, so it appears 13 times.) Give proof of your answer.

1) (6 pts) The input into S-box 6 in DES is 100110. What is the output, expressed in decimal (a single value in between 0 and 15, inclusive)?

2) (8 pts) In the middle of a round of DES, the input to the P array is 3D91AB75, in hexadecimal. What is the corresponding output from the P array, expressed in hexadecimal?

1) (16 pts) If the input to the S-boxes is 69E4 CF08 3B52, in hex, what is the output?

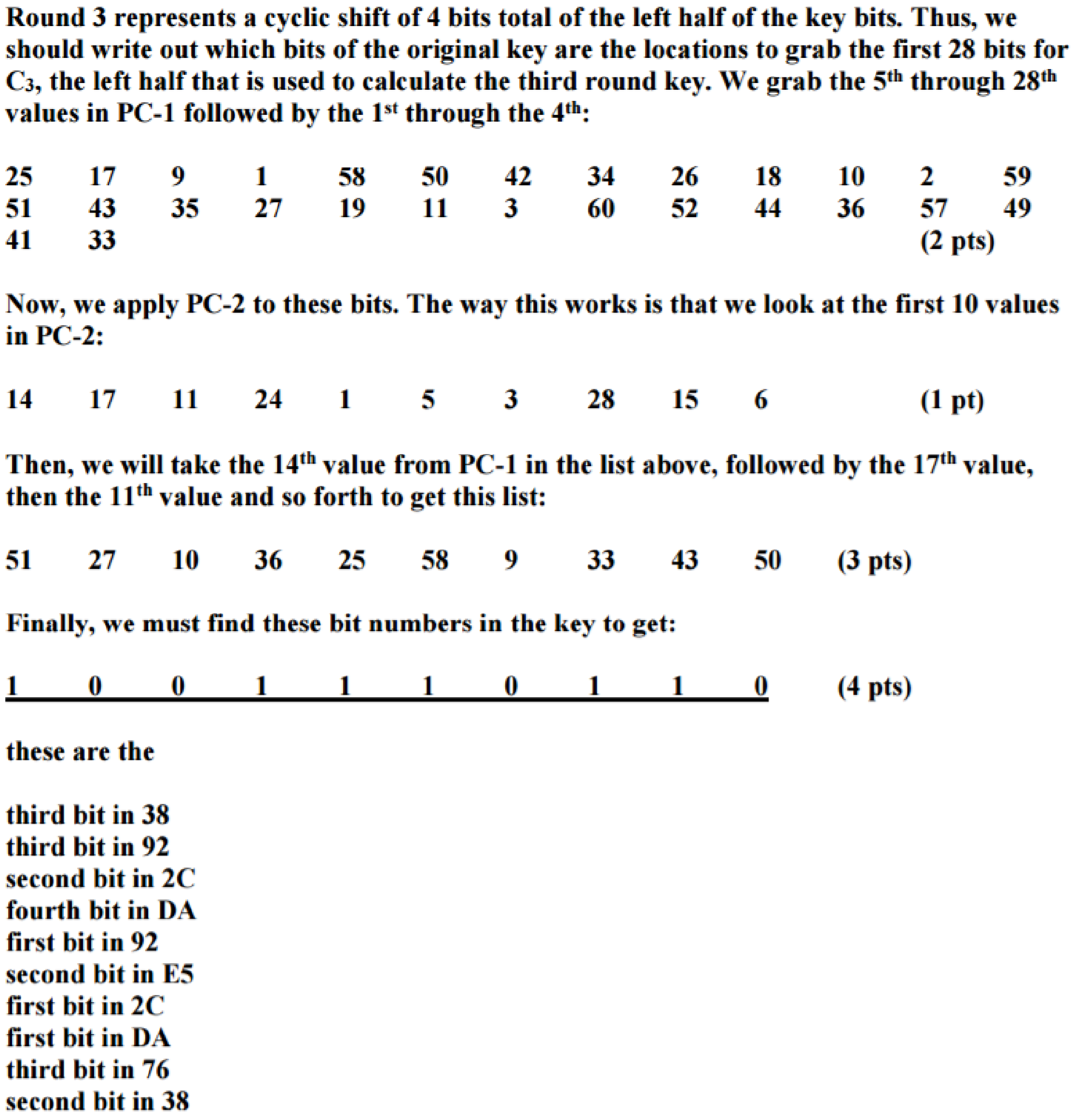


2) (8 pts) Imagine a DES-like cipher with a block size of 16 with the following IP matrix:

$$\begin{pmatrix} 6 & 13 & 7 & 5 \\ 11 & 15 & 9 & 16 \\ 2 & 14 & 3 & 12 \\ 8 & 1 & 4 & 10 \end{pmatrix}$$

What is the corresponding IP^{-1} matrix?

6) (10 pts) Let a DES key with parity bits expressed in HEX be BF 2C 57 92 DA 76 38 E5. Determine the first ten bits of the round 3 key.



Round 3 represents a cyclic shift of 4 bits total of the left half of the key bits. Thus, we should write out which bits of the original key are the locations to grab the first 28 bits for C_3, the left half that is used to calculate the third round key. We grab the 5th through 28th values in PC-1 followed by the 1st through the 4th:

25 17 9 1 58 50 42 34 26 18 10 2 59
51 43 35 27 19 11 3 60 52 44 36 57 49
41 33 (2 pts)

Now, we apply PC-2 to these bits. The way this works is that we look at the first 10 values in PC-2:

14 17 11 24 1 5 3 28 15 6 (1 pt)

Then, we will take the 14th value from PC-1 in the list above, followed by the 17th value, then the 11th value and so forth to get this list:

51 27 10 36 25 58 9 33 43 50 (3 pts)

Finally, we must find these bit numbers in the key to get:

1 0 0 1 1 1 0 1 1 0 (4 pts)

these are the

third bit in 38
third bit in 92
second bit in 2C
fourth bit in DA
first bit in 92
second bit in E5
first bit in 2C
first bit in DA
third bit in 76
second bit in 38

4) (10 pts) If the 48 bit input to the S-boxes in DES is 3A29B1234FE6, what is the 32 bit output, expressed in hexadecimal? Put a box around your final answer

Area F: AES

For 128-bit AES, the algorithm runs 10 rounds. For 192-bit AES, it runs 12 rounds. For 256-bit AES, it runs 14 rounds. Only the 128-bit AES will be discussed here in detail.

Assume that we have a block of 128 bits, split into 16 bytes, labeled **$a_{0,0}$, $a_{1,0}$, $a_{2,0}$, $a_{3,0}$, $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, $a_{0,2}$, $a_{1,2}$, $a_{2,2}$, $a_{3,2}$, $a_{0,3}$, $a_{1,3}$, $a_{2,3}$,** and **$a_{3,3}$.** You can visualize these 16 bytes filling up four columns of four bytes, with the first four elements in the first column, the second four elements in the second column, etc.

$a_{0,0}$	**$a_{0,1}$**	**$a_{0,2}$**	**$a_{0,3}$**
$a_{1,0}$	**$a_{1,1}$**	**$a_{1,2}$**	**$a_{1,3}$**
$a_{2,0}$	**$a_{2,1}$**	**$a_{2,2}$**	**$a_{2,3}$**
$a_{3,0}$	**$a_{3,1}$**	**$a_{3,2}$**	**$a_{3,3}$**

The following is repeated for 10 rounds:

Substitute bytes - For each of the sixteen bytes, look up their substitute in the S-box substitution chart, creating the new state matrix $b_{0,0}$, $b_{1,0}$, $b_{2,0}$, $b_{3,0}$, $b_{0,1}$, $b_{1,1}$, $b_{2,1}$, $b_{3,1}$, $b_{0,2}$, $b_{1,2}$, $b_{2,2}$, $b_{3,2}$, $b_{0,3}$, $b_{1,3}$, $b_{2,3}$, and $b_{3,3}$. The first character in the byte refers to the row in the S-box and the second character in the byte refers to the column.

Example:

A1	**03**	**98**	**B3**
C4	**D9**	**03**	**02**
44	**54**	**32**	**FE**
E3	**3D**	**7F**	**2A**

becomes...

32	**7B**	**46**	**6D**
1C	**35**	**7B**	**77**
1B	**20**	**23**	**BB**
11	**27**	**D2**	**E5**

after looking at the given S-box for AES:
http://www.cs.ucf.edu/courses/cis3362/fall2016/refsheets/AES-Tables.pdf

When decrypting, we use the inverse S-box, also given in the link above. The process is the same as encryption.

Add Round Key - Key Expansion

Pseudocode

```
KeyExpansion(byte key[16], word w[44]) {

  word temp;

  for (i=0; i<4; i++)
          w[i] = (key[4i], key[4i+1], key[4i+2], key[4i+3]);

  for (i=4; i<44; i++) {
          temp = w[i-1];
          if (i%4 == 0)
       temp = SubWord(RotWord(temp)) XOR Rcon[i/4];
          w[i] = w[i-4] XOR temp;
  }
}
```

Steps with examples:
Scenario: Let the round 3 key in AES be **01234567 89ABCDEF FEDBCA98 76543210** in hex. What will the first four bytes of the round 4 key be, represented in hex?

- Set temp based on first 8 bits of round key from the previous round:
  - Temp = 76543210
- Complete a left cyclic rotation (left shift of 2 bits):
  - rotWord(temp) = 54321076
- Complete s-box substitution (s-boxes given on reference sheet)
  - subWord(rotWord[temp]) = 2023CA38
- Get rcon[i] from table (i/4,if necessary) and pad with 0, where i is equal to what round you are currently on :
  - i = 4 ==04000000
- Perform XOR operation on previous 2 lines, in this case 2023CA30 and 04000000. The result will be set as your new temp:

    - Temp = 2023CA38⊕ 04000000 = 2423CA38
- Take the last 8 bits (w[i-4]):
    - 01234567
- And XOR with temp to get the final answer:
    - 2423CA38 ⊕01234567 = **25008F5F**

Example #2
Scenario: Consider the AES key schedule where we have
W[20] = 01234567
W[23] = 89abcdef
Expressed in hex. Calculate w[24].

Temp = 89abcdef
rotWord(temp) = abcdef89
subWord(rotWord[temp]) = 62bddfa7
rcon[24/4] = 20000000
XOR = 42bddfa7
W[i-4] = 01234567
XOR = **439e9ac0 - Final Answer**

ShiftRows Transformation - Arranges the state in a matrix and performs a circular shift for each row
Ezpz example: we have a 4x4 matrix arranged with bytes 1 2 3 … 16, and the matrix is **formed vertically** top down but **shifted horizontally**

1 5 9 13
2 6 10 14

3 7 11 15
4 8 12 16 Each row is shifted to the right based on the row #, **starting with 0**
SO NEW MATRIX BECOMES
1 5 9 13 (shifted 0)
14 2 6 10 (shifted 1)
11 15 3 7 (shifted 2)
8 12 16 4 (shifted 3) http://www.adamberent.com/documents/AESbyExample.pdf

----

MixColumns Transformation: For mixColumns place the standard matrix first and then the data matrix. Select the needed row from the standard matrix and select the column needed from the data matrix. In case of row 4 column 1, take the 4th row of the standard matrix, take the first column of the data matrix and do the following:

The leftmost element times the top element, second from left times second from top, and so on. Convert all hex to binary. Multiplying by 1 will leave the same value. To multiply by 2 add a zero at the end, if the leftmost bit is zero you are done (ignore the leftmost zero), if the leftmost bit is 1, remove it and xor the rest by 11011 (always the same), the result of the xor will be the answer. To multiply by 3, take the binary and multiply by 1, then take the same binary and multiply by 2, xor the results of both operations to get the needed binary. In order to get the final result take all 4 results from the computations above and xor them. Convert the final binary back to hex.

Example on the following page:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} EA & 95 & A2 & 12 \\ 2C & 7E & 36 & 4F \\ 97 & F9 & 20 & 62 \\ B2 & C8 & A7 & D3 \end{bmatrix}$$

row 4 col 1

```
03 x EA = 03 x 1110 1010 = 01 x 11101010 ⊕
                           02 x 11101010

01 x 11101010 = 11101010
02 x 11101010 = 111010100 = 11010100 ⊕
                            ↓↓↓11011
                            11001111

⊕ 11101010
  11001111
  00100101

01 x 2C = 01 x 00101100 = 00101100

01 x 97 = 01 x 10010111 = 10010111

02 x B2 = 02 x 10110010 = 101100100 = 01100100 ⊕
                                      ↓↓↓11011
                                      01111111

00100101
00101100
10010111
01111111
11100001
```

$\boxed{\text{row 4 col 1} = 11100001 = E1}$

RSA

RSA is one of the first practical public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret. In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1977. Clifford Cocks, an English mathematician working for the UK intelligence agency GCHQ, had developed an equivalent system in 1973, but it was not declassified until 1997.[1]

A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime numbers can feasibly decode the message.[2] Breaking RSA encryption is known as the RSA problem; whether it is as hard as the factoring problem remains an open question.

RSA is a relatively slow algorithm, and because of this it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed.

Example Problem:

Consider an RSA system with $n = 91$ and $e = 25$. Calculate d.

$n = 91 = 7 \times 13$, so $\varphi(n) = \varphi(7 \times 13) = \varphi(7) \times \varphi(13) = 6 \times 12 = 72$.

$d = e-1 \bmod \varphi(n)$. Thus, $d = 25 -1 \bmod 72$.

Use the Extended Euclidean Algorithm to find d:

$72 = 2 \times 25 + 22$

$25 = 1 \times 22 + 3$

$22 = 7 \times 3 + 1$

$22 - 7 \times 3 = 1$

$22 - 7(25 - 22) = 1$

$22 - 7 \times 25 + 7 \times 22 = 1$

$8 \times 22 - 7 \times 25 = 1$

$8(72 - 2 \times 25) - 7 \times 25 = 1$

$8 \times 72 - 16 \times 25 - 7 \times 25 = 1$

$8 \times 72 - 23 \times 25 = 1$

It follows that $d \equiv -23 \equiv 49 \pmod{72}$

Knapsack Cipher

The Knapsack Cryptosystem was made by Merkle and Hellman, which was was based off of the “knapsack problem” : Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

There is a subset{}, and a chosen target that numbers from the subset add up to.

In the Knapsack Cryptosystem, The subset chosen is “super-increasing” which means every number in the subset is greater than the sum of all numbers beforehand.

$$s_{n+1} > \sum_{j=1}^{n} s_j$$

Super-Increasing subset representation

To create the public key, we must take the values and multiply by a number a (which has no factors common with any number in the subset, and take a modulus (p) of a value which is greater than the total sum of the super-increasing set.

The public key will end up looking like:

({a(s1)modm, a(s2)modp, a(s3)modp, a(s4)modp, a(s5)modp, …}a p)

While the private key ends up being the super-increasing subset

{(s1, s2, s3, s4, s5, …)}

Encrypting a message is a simple as taking in the amount of bits as the number of entries in the subset, using them to tell which numbers to use in the public key to add together, producing a part of the encrypted message.

For example, if you had the binary 10110, you would have to add together

a(s1)modp + a(s3)modp + a(s4)modp = c

where c = part of encrypted message sent

To Decrypt a message, the receiver has to have the prime ‘a’, and the modulus used ‘p’. The receiver would then find the inverse modulo of a mod p, ‘a^-1 modp’ and multiply that by the received values mod p a’ = inverse modulo amodp

a’ * c mod p

This value is compared to the superincreasing private key as a sort of ‘Target’ to pick out which numbers from the subset are used to add up to the decoded message.

Here is a very thorough example

Plaintext 111010101101111001

Private key {1, 2, 4, 10, 20, 40}

Calculating Public key

1×53 mod(120) = 53

2×53 mod(120) = 106

4×53 mod(120) = 92

10×53 mod(120) = 50

20×53 mod(120) = 100

40×53 mod(120) = 80

Public key {53, 106, 92, 50, 100, 80}

Now we take the plaintext in groups of 6 (because that's how many numbers are in the Public Key subset) adding the numbers from subset that are evaluating to 1, leaving the 0's out.

s1 + s2 + s3 + s5 = c (from public key) only for first block of bits

111010 = 53 + 106 + 92 + 100 = 351

101101 = 53+ 92 + 50 + 80 = 275

111001 = 53 + 106 + 92 + 80 = 331

Cipher text to send is 351 275 331

Receiver finds inverse 53 (a) mod 120 (p) which is 77

77 is now taken and multiplied by the ciphertext and modded 'p'

351×77 mod(120) = 27 = 111010 (1+2+4+20) compared to private key

275×77 mod(120) = 55 = 101101

331×77 mod(120) = 47 = 111001

Decoding the message to 111010101101111001

Example from Arup Guha’s slides Week 11

Plaintext: 1001

Private key {1, 5, 7 19}

Modulo ‘p’ = 41

Prime ‘a’ = 10

Public key {10, 9, 29, 26}

Plain = 1 0 0 1

Cipher = 10 + 26 = 36

36 is sent.

Receiver (has ‘a’ and ‘p’) calculates a (inverse) mod p

10 * 37 mod 41 = 1

And 37 (‘a’ inverse mod ‘p’) is multiplied by 36 (plaintext) mod ‘p’

37 * 36 mod 41

20mod41

Which evaluates back out to

1 0 0 1

1 + 19 = 20

Elliptic Curve Cryptography

Prepared by Group 7: Austin Shipley, Isaac Ehlers, David Dill, and Chey Smith

TL;DR

Elliptic curves are like this:

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

To do P + Q = R, you use this:

$$\Delta = (y_q - y_p) / (x_q - x_p)$$

$$x_r = \Delta^2 - x_p - x_q$$

$$y_r = \Delta*(x_p - x_r) - y_p$$

If you want to add something to itself like P + P = R, use this:

$$\Delta = (3x_p^2 + a) / (2y_p)$$

$$x_r = \Delta^2 - 2x_p$$

$$y_r = \Delta * (x_p - x_r) - y_p$$

gg, ez.

In more detail on the next page

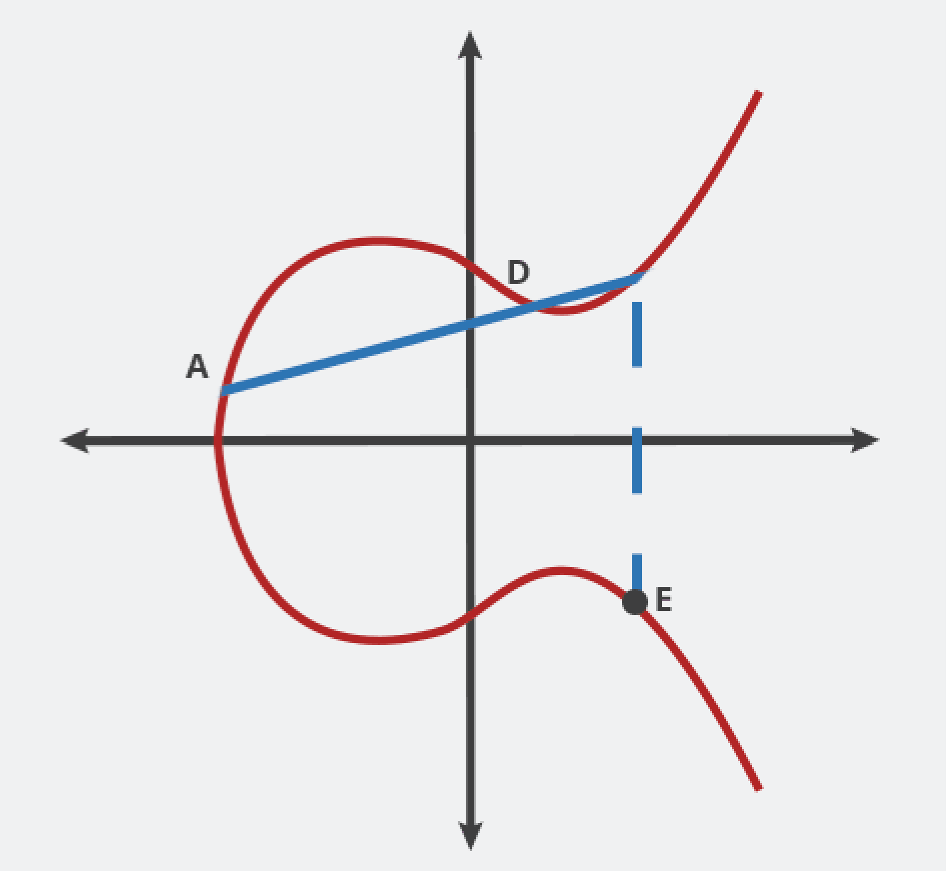
Elliptic curve arithmetic is strange, but useful, and it helps keep us secure by introducing a new type of problem: the elliptic curve discrete logarithm problem.

Elliptic curves are defined as the set of points that form a solution to an equation of the form:

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

Which is often reduced to the notation $E_p(a,b)$

When you add two points on an elliptic curve, you draw a line between them and take the opposite point of where that line intersects the curve. As seen in the figure.



Pretty simple, and we have a formula for it! (listed at the beginning)

What’s useful about Elliptic curve arithmetic is that repeated operations yield difficult to predict answers, in fact the only way to arrive at the answer is to perform all the calculations that led up to it. In regular arithmetic 2 + 2 = 4, 4 + 2 = 6, 6 + 2 = 8 can be simplified to 2*4, there is no simplification in EC arithmetic. If you want to calculate A + A = B, A + B = C, A + C = D you have to perform each and every calculation, in sequence. This parallels the elliptic curve discrete logarithm problem mentioned before, and makes it an excellent candidate for a cryptographic system.

Here's how it can be used to generate a shared secret key:

Alice and Bob decide on a curve to use and pick an arbitrary point G on the curve.

Alice generates a random integer aa which is her private key, and finds her public key A=aa*G (i.e. G added to itself aa times).

Bob uses his random private key bb to form B=bb*G.

They exchange public ECC keys, and then Alice finds their shared secret by calculating aa*B, and Bob finds bb*A

Since aa*B=aa*(bb*G)=(aa*bb)*G=(bb*aa)*G=bb*(aa*G)=bb*A, they've computed the same value.

Example

Given the curve $E_{23}(14, 11)$ calculate A + B and 2A where A = (6, 9) and B = (3, 7):

(Remember this is in mod 23, so no division, instead multiply by the inverse)

A + B:

$\Delta = (7 – 9) / (3 – 6) = -2 * -3^{-1} = -2 * 15 = -30 = 16$

$x_C = 16^2 – 6 – 3 = 247 = 17$

$y_C = 16 * (6 – 17) – 9 = -185 = 22$

A + B = (17, 22)

2A:

$\Delta = (3*6^2 + 14) / (2*9) = 122 * 18^{-1} = 7 * 9 = 63 = 17$

$x_r = 17^2 – 2*6 = 288 = 1$

$y_r = 17 * (6 – 1) – 9 = 76 = 7$

2A = (1, 7)

David Dill
Isaac Ehlers
Austin Shipley
Chey Smith

Group 7: Elliptic Curve Cryptography

Group Theory:

Elliptic curve arithmetic is a specific part of Group Theory in mathematics. It falls under the stricter category of Abelian Groups. Regular groups require four of the five following identities (A1 through A4) to be considered a group, whereas Abelian Groups require a fifth, more specific identity (A5):

<u>(Abelian) Group Properties:</u>

A1. "Closure" - If *a* is an element of group *G* and *b* is an element of group *G*, then the dot product of *a* and *b* is also an element in group *G*:

$$a \,\varepsilon\, G \,,\, b \,\varepsilon\, G \;\square\; a \circ b \,\varepsilon\, G$$

A2. "Associative" - The following equivalence must be valid, where a, b, and *c* are all elements within some group *G* (*a*, *b*, *c* ε *G*):

$$a \circ (b \circ c) = (a \circ b) \circ c$$

A3: "Identity Element" - There must exist elements *e* and *a* within some group *G* (∃*e*, *a* ε *G*) such that:

$$a \circ e = e \circ a = a$$

A4: "Inverse Element" - For each element *a* within some group *G*, there must exist an element *a'* also within *G* (∀*a* ε *G*, ∃*a'* ε *G*) such that:

$$a \circ a' = e,\; a' \circ a = e$$

A5: "Commutative" - For elements *a* and *b* within some group *G* (*a*, , the following must be true:

$$a \circ b = b \circ a$$

David Dill
Isaac Ehlers
Austin Shipley
Chey Smith

Elliptic Curve Arithmetic:

Elliptic Curve Cryptography (ECC) is a form of public key cryptography that utilizes the properties elliptic curves and the points that lie on them. Points on the elliptic curves used in ECC must satisfy the following equation (and it is these points that make up the elements of the group defined above):

$$\mathrm{E}_q(a, b)\text{: } (y^2 \equiv x^3 + ax + b) \bmod q,\ q\ \varepsilon\ \Pi \text{ (prime)}$$

When executing Elliptic Curve Cryptography, there are two operations that must be performed on the curve's points (P_n) - elliptic curve "addition" of points with differing *x*-coordinates (determining $P_p + P_q$) and elliptic curve "addition" of points with equal *x*-coordinates (specifically, determining $2P_n$):

<u>Addition ($x_p \neq x_q$):</u>

If we have two points of the form (x_p, y_p) and (x_q, y_q) and wish to add them to determine the resulting point (x_r, y_r), we can use the following steps:

1. Determine the slope delta:
   $$\Delta = \frac{y_q - y_p}{x_q - x_p}$$
2. Use delta to determine the *x*-coordinate of the resulting point:
   $$x_r = \Delta^2 - x_p - x_q$$
3. Now, using the resulting *x*-coordinate, determine the *y*-coordinate of the resulting point:
   $$y_r = \Delta(x_p - x_r) - y_p$$

Note: This will only work if $x_p \neq x_q$. Or, in other words, the slope between the points being added is *defined*.

<u>Addition ($x_p = x_q$):</u>

Similarly, if we have two points of the form (x_p, y_p) and (x_p, y_q) - note the *x*-coordinates are the same, thus the slope between them is *undefined* - and wish to add them to determine the resulting point (x_r, y_r), we can use the following steps:

1. Determine delta:
   $$\Delta = \frac{3{x_p}^2 + a}{2y_p}$$
2. Use delta to the *x*-coordinate of the resulting point:
   $$x_r = \Delta^2 - 2x_p$$
3. Now, using the resulting *x*-coordinate, determine the *y*-coordinate of the resulting point:
   $$y_r = \Delta(x_p - x_r) - y_p$$

You now have the elements necessary to construct the resulting point(s) (x_r, y_r).

Elliptic Curve Cryptography:

David Dill
Isaac Ehlers
Austin Shipley
Chey Smith

When setting up a cryptosystem that utilizes elliptic curve arithmetic, you must determine the following:

Global Public Elements:

1. The elliptic curve with notation $E_q(a, b)$ as shown above.
2. Point G, a point on the curve with "large order" - takes a long time to cycle back around to where you started (typically considered a generator of prime q).

User Elements:

User A (Alice):

1. Public - some integer n_A, where: $n_A < q$
2. Private - some integer P_A, where: $P_A = n_A \times G$

User B (Bob):

1. Public - some integer n_B, where: $n_B < q$
2. Private - some integer P_B, where: $P_B = n_B \times G$

From here, Alice can calculate K = $n_A \times P_B$, and Bob can calculate K = $n_B \times P_A$. Note that both of these values come out to the same value of K = $n_A \times n_B \times G$ (similar to the process seen in the Diffie-Hellman key exchange).

Now, for Alice to send message P_m to Bob:

1. Alice must choose a random integer k and send the following ciphertext to Bob:

   $C_m = \{kG, P_m + kP_B\}$
2. Bob receives C_m, and knowing n_B, can decrypt the ciphertext through the use of the following calculations:

$$\begin{aligned}(P_m + kP_B) - n_B(kG) &= (P_m + k(n_B G)) - n_B(kG) \\ &= P_m + (kn_B G - kn_B G) \\ &= P_m\end{aligned}$$

Thus, using only the information sent with C_m and his own private key n_B, Bob is able to decrypt Alice's message successfully.

CIS 3362
group 12

Birthday Attack Problem

One year on Venus is 225 days. What is the probability that of a random sample of 10 Venetians, all of them have different birthdays? Please write down the answer in product notation and then use your calculator to get an approximation for the value.

The probability that two selected Venetians have the a different birthday is $\frac{224}{225}$, because if one person has a birthday, there are 224 other days that the second birthday could be on. This means, to find the probability that none of the 10 random Venetians have the same birthday, you would need to calculate:

$$\frac{225}{225}*\frac{224}{225}*\frac{223}{225}*\frac{222}{225}*\frac{221}{225}*\frac{220}{225}*\frac{219}{225}*\frac{218}{225}*\frac{217}{225}*\frac{216}{225}*\frac{215}{225}$$

$$==$$

$$\prod_{n=0}^{10}\frac{225-n}{225}$$

$$or$$

$$\frac{225!}{(225-n)!\,255^n}\ , where\ n=10$$

This comes out to being $\approx \mathbf{0.8164\ or\ 81.64}\%$ chance that none of the Venetians have the same birthday.

A more generalized equation for a Birthday Probability problem would be:

$$P(n)=\prod_{t=0}^{n}\frac{d-t}{d}\ or\ \frac{d!}{(d-n)!\,d^n}$$

Where **d** = the number of days and **n** = the sample of birthdays.

If the question were asking for the probability that at least 2 have the same birthday, it would be:

1 – P(n)

In this case, 1 - .8164 = **.1836** or **18.36%** chance at least 2 Venetians in a group of 10 have the same birthday.

El Gamal Cryptosystem

Alice Creates:

Public	Private
p - Prime Number	X - Random number < p - 1
α - Primitive Root of p	
$Y = \alpha^X \bmod p$	

1. Plaintext M, $M < p$
2. Random integer k, $k < p$
3. $K = (Y)^k \bmod p$
4. $C_1 = \alpha^K$
5. $C_2 = KM \bmod p$

   $= Y^k * M \bmod p$

   $= (\alpha^X)^k * M \bmod p$

   $= \alpha^{Xk} * M \bmod p$
6. Send (C_1, C_2)

To Decrypt:

$K = C_1^X = \alpha^{k*X} \bmod p$

$M = K^{-1} * C_2 \bmod p$

$= K^{-1} * K * M \bmod p$

Example

Let $p = 11$, $\alpha = 2$

Let $X = 8$

$Y = \alpha^X$

$Y = 2^8 = 256 \bmod 11 = 3$ $\qquad Y = 3$

1. Let $M = 5$,
2. Let $k = 9$
3. $K = Y^k \bmod p = 3^9 \bmod 11 = 4$ $\qquad K = 4$
4. $C_1 = \alpha^k = 2^9 \bmod 11 = 6$
5. $C_2 = K*M = 4*5 = 20 \bmod 11 = 9$
6. Send $(C_1, C_2) = (6, 9)$

If decrypting:

$K = C_1^X$

$= \alpha^{k*X} \bmod p$

$= Y^k \bmod p$

$= 3^9 \bmod 11 = 11$

$M = K^{-1} * C_2 \bmod p = K^{-1} * K * M \bmod p$

$4^{-1} \bmod 11 = 3$

$4 * 3 * 5 \bmod 11 = 5 \bmod 11$

El Gamal Cryptosystem

Alice Creates:

Public	Private
p - Prime Number	X - Random number < p - 1
α - Primitive Root of p	
$Y = \alpha^{X} \bmod p$	

1. Plaintext M, $M < p$
2. Random integer k, $k < p$
3. $K = (Y)^{k} \bmod p$
4. $C_1 = \alpha^{K}$
5. $C_2 = KM \bmod p$
   $= Y^{k} * M \bmod p$
   $= (\alpha^{X})^{k} * M \bmod p$
   $= \alpha^{Xk} * M \bmod p$
6. Send (C_1, C_2)

To Decrypt:

$K = C_1^{X} = \alpha^{k*X} \bmod p$
$M = K^{-1}*C_2 \bmod p$
$= K^{-1} * K * M \bmod p$

Example

Let $p = 11$, $\alpha = 2$
Let $X = 8$
$Y = \alpha^{X}$
$Y = 2^{8} = 256 \bmod 11 = 3$ $Y = 3$

1. Let $M = 5$,
2. Let $k = 9$
3. $K = Y^{k} \bmod p = 3^{9} \bmod 11 = 4$ $K = 4$
4. $C_1 = \alpha^{k} = 2^{9} \bmod 11 = 6$
5. $C_2 = K*M = 4*5 = 20 \bmod 11 = 9$
6. Send $(C_1, C_2) = (6, 9)$

If decrypting:

$K = C_1^{X}$
$= \alpha^{k*X} \bmod p$
$= Y^{k} \bmod p$
$= 3^{9} \bmod 11 = 11$
$M = K^{-1} * C_2 \bmod p = K^{-1} * K * M \bmod p$
$4^{-1} \bmod 11 = 3$
$4 * 3 * 5 \bmod 11 = 5 \bmod 11$

Digital Signature Standard (DSS) Notes

Key Generation:

1. Choose a prime p:

   $2^{L-1} < p < 2^{L}$, where $512 \leq L \leq 1024$ (512 bits to 1024 bits)

2. Choose a prime q:

   $q \mid (p-1)$, where q is 160 bits

3. Choose a primitive root/generator g:

   $g = h^{(p-1)/q} \bmod p$, where $1 < h < p-1$

4. Compute private and public keys for a single user:
   a. Choose a random secret key x, where $0 < x < q$
   b. Calculate the public key:
      $y = g^{x} \bmod p$

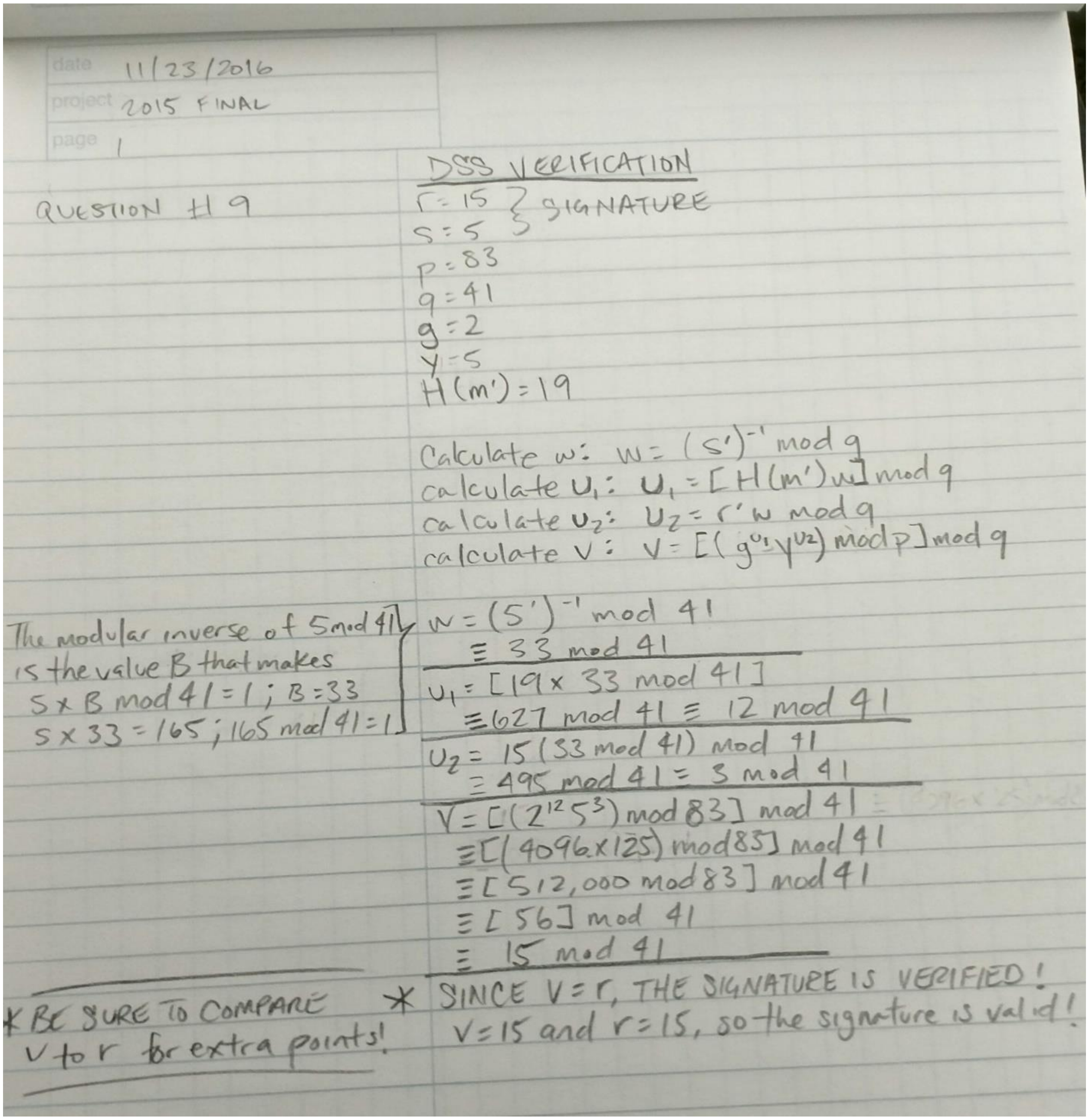
Signing:

1. Let H be the hashing function and m the message
2. Generate a random per-message value k, where $0 < k < q$
3. Calculate $r = (g^{k} \bmod p) \bmod q$
4. Calculate $s = [k^{-1} (H(m) + xr)] \bmod q$

   Signature: (r,s)

Verify:

1. Calculate $w = (s')^{-1} \bmod q$
2. Caculate $u_1 = [H(m')w] \bmod q$
3. Caculate $u_2 = r'w \bmod q$
4. Calculate $v = (g^{u1}y^{u2} \bmod p) \bmod q$
5. Signature is valid if $v = r$



date 11/23/2016
project 2015 FINAL
page 1

QUESTION #9

DSS VERIFICATION

$r = 15$ } SIGNATURE
$s = 5$
$p = 83$
$q = 41$
$g = 2$
$y = 5$
$H(m') = 19$

Calculate w: $w = (s')^{-1} \bmod q$
calculate u_1: $u_1 = [H(m')w] \bmod q$
calculate u_2: $u_2 = r'w \bmod q$
calculate v: $v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$

The modular inverse of 5 mod 41 is the value B that makes
$5 \times B \bmod 41 = 1$; $B = 33$
$5 \times 33 = 165$; $165 \bmod 41 = 1$

$w = (5')^{-1} \bmod 41$
$\equiv 33 \bmod 41$

$u_1 = [19 \times 33 \bmod 41]$
$\equiv 627 \bmod 41 \equiv 12 \bmod 41$

$u_2 = 15(33 \bmod 41) \bmod 41$
$\equiv 495 \bmod 41 \equiv 3 \bmod 41$

$v = [(2^{12} 5^3) \bmod 83] \bmod 41$
$\equiv [(4096 \times 125) \bmod 83] \bmod 41$
$\equiv [512{,}000 \bmod 83] \bmod 41$
$\equiv [56] \bmod 41$
$\equiv 15 \bmod 41$

* SINCE $v = r$, THE SIGNATURE IS VERIFIED!
$v = 15$ and $r = 15$, so the signature is valid!

* BE SURE TO COMPARE v to r for extra points!