Lab08 - Block Ciphers, Meet in the Middle using DES and Baby Rinjdael

Part 1

DES uses 56 cryptographic bits in a key, meaning there are 2^{56} possible unique values in a key. While it may have been computationally sound back in the day, computers eventually advanced to the point that DES could be brute forced a little too easily. One might have the bright idea to use DES twice, once to encrypt the plaintext, and then another time to encrypt the ciphertext with another key. In theory, this would require a brute force to take $2^{56} * 2^{56} = 2^{112}$ attempts to crack DES used twice. In practice, it only requires $2^{56} + 2^{56} = 2^{57}$ attempts to crack DES used twice. How many times less secure is this? Include this in your lab report

For example, (This is for reference only) say we are given the following information:

knownPlaintext = `"ILikeDogs"`
k1 = `"1287abf7d6380c _ _"`
k2 = `"87a6ff3eec3781 _ _"`
ciphertext = `"urrUYvAyw/GZu4pWU4AE7A1ruD6fmmwVdNn6ys1agTw="`
secretMSG = `“l9Z3PKBq0w/QBSMjEmOuenGqyrKkk5+OWhL3nU/zA5s=”`

We can write code to brute force k1 and k2. If the encryption from the known plaintext and k1 is the same as decrypting the ciphertext with k2 we have found the keys used to encrypt the ciphertext and secret message. The output from your program might look like the figure below:

```
cpre331@desktop:~$ python3 Lab8.py
Trying k1=1287abf7d6380c00 and k2=87a6ff3eec378100
Trying k1=1287abf7d6380c00 and k2=87a6ff3eec378101
Trying k1=1287abf7d6380c00 and k2=87a6ff3eec378102
Trying k1=1287abf7d6380c00 and k2=87a6ff3eec378103
…
…
…
Trying k1=1287abf7d6380c04 and k2=87a6ff3eec378138
Trying k1=1287abf7d6380c04 and k2=87a6ff3eec378139
Trying k1=1287abf7d6380c04 and k2=87a6ff3eec37813a
*** MATCH FOUND ***
```

```
Key1: 1287abf7d6380c04
Key2: 87a6ff3eec37813a
```

Now that we have k1 and k2, we can decrypt any other messages that were encrypted with the same key values, such as the secret message. An example of how to do this can be seen below and in the code template. The secret message is “DontUse2XDES”

Template code: scp repo@repo.331.com:/home/repo/lab08/* .

```
cpre331@desktop:~$ /bin/echo -n l9Z3PKBq0w/QBSMjEmOuenGqyrKkk5+OWhL3nU/zA5s= | openssl enc -d -a -des-cbc -nosalt -iv 0000000000000000 -A -K 87a6ff3eec37813a | xxd
00000000: 714f 6b78 4148 4d6e 3939 5139 454b 3143  qOkxAHMn99Q9EK1C
00000010: 6552 546d 3351 3d3d                      eRTm3Q==

cpre331@desktop:~$  /bin/echo -n qOkxAHMn99Q9EK1CeRTm3Q== | openssl enc -d -a -des-cbc -nosalt -iv 0000000000000000 -A -K 1287abf7d6380c04 | xxd
00000000: 446f 6e74 5573 6532 5844 4553            DontUse2XDES
```

This information is what you will work with in this lab. Submit your functional code on canvas. What is k1 and k2?

```
knownPlaintext  = "Tatooine"
k1 = "319df2f409baee _ _"
k2 = "64abc398ac4fee _ _"
ciphertext = "tm97RIBRG3eY8fkb0iU696gHGhfGxnYZGVcB2sJRDK4="
secretMSG =
“eM+KmLs+5TFQEknuRX2fzhigcZPCkSho7bf/73mh8bFaHaCFLW7AoQ==”
```

Part 2

A Iowa State University graduate student in conjunction with Dr. Cliff Bergman defined a Baby Rinjdael cipher that has been used for several evaluations of the security of AES. It is a much smaller version of the cipher, but it uses similar constructs as the fully functional Rinjdael.

In lab you will code Baby Rinjdael using the following specifications the graduate student defined.

Block size: 16 bits which are defined as 4 hexadecimal digits $h_0h_1h_2h_3$

Note that h_0 consists of the first four bits of the input stream. However, when h_0 is considered as a hex digit, the first bit is considered the high-order bit.
For example, the input block 1000 1100 0111 0001 would be represented with
h_0 =8, h_1 = c, h_2 = 7, h_3 = 1.

Key size: Also 16 bits represented as 4 hexadecimal digits k_0,k_1,k_2,k_3 The key scheduling algorithm is complex so for the purposes of CprE/CybE 331 we will use one key and its resulting key schedule (round keys).

State: The state is usually considered to be a 2×2 array of hex digits. However, for the mixcolumns, the state is considered to be an 8×2 array of bits. In converting between the two, each hex digit is considered to be a column of 4 bits with the high-order bit at the top.
The input block is loaded into the state by mapping $h_0h_1h_2h_3$ (8c71) as

h_0	h_2
h_1	h_3

8	7
c	1

To write this as the 8x2 matrix the input block 1000 1100 0111 0001 would be loaded as

1	0
0	1
0	1
0	1
1	0
1	0
0	0
0	1

Number of rounds: The default is 4 which is what we will use for this lab.

SubByte: This is a simple lookup table where the lookup is applied to each hexadecimal digit of that state

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
s(x)	a	4	3	b	8	e	2	c	5	7	8	f	0	1	9	d

As an example

$$\begin{array}{cc} 8 & 7 \\ c & 1 \end{array} \quad \rightarrow \quad \begin{array}{cc} 5 & c \\ 0 & 4 \end{array}$$

ShiftRow: Again, this function has been simplified greatly. The entries in the second row of the state are swapped.

$$\begin{array}{cc} h_0 & h_2 \\ h_1 & h_3 \end{array} \quad \rightarrow \quad \begin{array}{cc} h_0 & h_2 \\ h_3 & h_1 \end{array}$$

MixColumn: The matrix t is an 8 x 8 matrix of bits.

$$\begin{array}{cccccccc} 1&0&1&0&0&0&1&1 \\ 1&1&0&1&0&0&0&1 \\ 1&1&1&0&1&0&0&0 \\ 0&1&0&1&0&1&1&1 \\ 0&0&1&1&1&0&1&0 \\ 0&0&0&1&1&1&0&1 \\ 1&0&0&0&1&1&1&0 \\ 0&1&1&1&0&1&0&1 \end{array} \qquad \begin{array}{cccccccc} t_{00}&t_{01}&t_{02}&t_{03}&t_{04}&t_{05}&t_{06}&t_{07} \\ t_{10}&t_{11}&t_{12}&t_{13}&t_{14}&t_{15}&t_{16}&t_{17} \\ t_{20}&t_{21}&t_{22}&t_{23}&t_{24}&t_{25}&t_{26}&t_{27} \\ t_{30}&t_{31}&t_{32}&t_{33}&t_{34}&t_{35}&t_{36}&t_{37} \\ t_{40}&t_{41}&t_{42}&t_{43}&t_{44}&t_{45}&t_{46}&t_{47} \\ t_{50}&t_{51}&t_{52}&t_{53}&t_{54}&t_{55}&t_{56}&t_{57} \\ t_{60}&t_{61}&t_{62}&t_{63}&t_{64}&t_{65}&t_{66}&t_{67} \\ t_{70}&t_{71}&t_{72}&t_{73}&t_{74}&t_{75}&t_{76}&t_{77} \end{array}$$

To calculate the MixColumn, the state is thought of as an 8x2 matrix of bits.

$$\begin{array}{cc} h_{00} & h_{20} \\ h_{01} & h_{21} \\ h_{02} & h_{22} \\ h_{03} & h_{23} \\ h_{10} & h_{30} \\ h_{11} & h_{31} \\ h_{12} & h_{32} \\ h_{13} & h_{33} \end{array}$$

Use matrix multiplication modulo 2 to calculate the new values of the 8 x 2 bit matrix. The first column used against all the rows in the t table are the values in h_0 and h_1 which produce new h_0 and h_1 values as shown in the table and equations below.

h_{00}	h_{20}
h_{01}	h_{21}
h_{02}	h_{22}
h_{03}	h_{23}
h_{10}	h_{30}
h_{11}	h_{31}
h_{12}	h_{32}
h_{13}	h_{33}

t_{00}	t_{01}	t_{02}	t_{03}	t_{04}	t_{05}	t_{06}	t_{07}
t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}
t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}
t_{30}	t_{31}	t_{32}	t_{33}	t_{34}	t_{35}	t_{36}	t_{37}
t_{40}	t_{41}	t_{42}	t_{43}	t_{44}	t_{45}	t_{46}	t_{47}
t_{50}	t_{51}	t_{52}	t_{53}	t_{54}	t_{55}	t_{56}	t_{57}
t_{60}	t_{61}	t_{62}	t_{63}	t_{64}	t_{65}	t_{66}	t_{67}
t_{70}	t_{71}	t_{72}	t_{73}	t_{74}	t_{75}	t_{76}	t_{77}

New h_{00} =
$((h_{00} * t_{00}) + (h_{01} * t_{01}) + (h_{02} * t_{02}) + (h_{03} * t_{03}) + (h_{10} * t_{04}) + (h_{11} * t_{05}) + (h_{12} * t_{06}) + (h_{13} * t_{07})) \% 2$

New h_{01} =
$((h_{00} * t_{10}) + (h_{01} * t_{11}) + (h_{02} * t_{12}) + (h_{03} * t_{13}) + (h_{10} * t_{14}) + (h_{11} * t_{15}) + (h_{12} * t_{16}) + (h_{13} * t_{17})) \% 2$

New h_{02} =
$((h_{00} * t_{20}) + (h_{01} * t_{21}) + (h_{02} * t_{22}) + (h_{03} * t_{23}) + (h_{10} * t_{24}) + (h_{11} * t_{25}) + (h_{12} * t_{26}) + (h_{13} * t_{27})) \% 2$

New h_{03} =
$((h_{00} * t_{30}) + (h_{01} * t_{31}) + (h_{02} * t_{32}) + (h_{03} * t_{33}) + (h_{10} * t_{34}) + (h_{11} * t_{35}) + (h_{12} * t_{36}) + (h_{13} * t_{37})) \% 2$

New h_{10} =
$((h_{00} * t_{40}) + (h_{01} * t_{41}) + (h_{02} * t_{42}) + (h_{03} * t_{43}) + (h_{10} * t_{44}) + (h_{11} * t_{45}) + (h_{12} * t_{46}) + (h_{13} * t_{47})) \% 2$

New h_{11} =
$((h_{00} * t_{50}) + (h_{01} * t_{51}) + (h_{02} * t_{52}) + (h_{03} * t_{53}) + (h_{10} * t_{54}) + (h_{11} * t_{55}) + (h_{12} * t_{56}) + (h_{13} * t_{57})) \% 2$

New h_{12} =
$((h_{00} * t_{60}) + (h_{01} * t_{61}) + (h_{02} * t_{62}) + (h_{03} * t_{63}) + (h_{10} * t_{64}) + (h_{11} * t_{65}) + (h_{12} * t_{66}) + (h_{13} * t_{67})) \% 2$

New h_{13} =
$((h_{00} * t_{70}) + (h_{01} * t_{71}) + (h_{02} * t_{72}) + (h_{03} * t_{73}) + (h_{10} * t_{74}) + (h_{11} * t_{75}) + (h_{12} * t_{76}) + (h_{13} * t_{77})) \% 2$

The second column used against all the rows in the t table are the values in h_2 and h_3 which produce new h_2 and h_3 values as shown in the table and equations below.

h_{00}	h_{20}	t_{00}	t_{01}	t_{02}	t_{03}	t_{04}	t_{05}	t_{06}	t_{07}
h_{01}	h_{21}	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}
h_{02}	h_{22}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}
h_{03}	h_{23}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}	t_{35}	t_{36}	t_{37}
h_{10}	h_{30}	t_{40}	t_{41}	t_{42}	t_{43}	t_{44}	t_{45}	t_{46}	t_{47}
h_{11}	h_{31}	t_{50}	t_{51}	t_{52}	t_{53}	t_{54}	t_{55}	t_{56}	t_{57}
h_{12}	h_{32}	t_{60}	t_{61}	t_{62}	t_{63}	t_{64}	t_{65}	t_{66}	t_{67}
h_{13}	h_{33}	t_{70}	t_{71}	t_{72}	t_{73}	t_{74}	t_{75}	t_{76}	t_{77}

New h_{20} =
$((h_{20} * t_{00}) + (h_{21} * t_{01}) + (h_{22} * t_{02}) + (h_{23} * t_{03}) + (h_{30} * t_{04}) + (h_{31} * t_{05}) + (h_{32} * t_{06}) + (h_{33} * t_{07}))\ \%2$

New h_{21} =
$((h_{20} * t_{10}) + (h_{21} * t_{11}) + (h_{22} * t_{12}) + (h_{23} * t_{13}) + (h_{30} * t_{14}) + (h_{31} * t_{15}) + (h_{32} * t_{16}) + (h_{33} * t_{17}))\ \%2$

New h_{22} =
$((h_{20} * t_{20}) + (h_{21} * t_{21}) + (h_{22} * t_{22}) + (h_{23} * t_{23}) + (h_{30} * t_{24}) + (h_{31} * t_{25}) + (h_{32} * t_{26}) + (h_{33} * t_{27}))\ \%2$

New h_{23} =
$((h_{20} * t_{30}) + (h_{21} * t_{31}) + (h_{22} * t_{32}) + (h_{23} * t_{33}) + (h_{30} * t_{34}) + (h_{31} * t_{35}) + (h_{32} * t_{36}) + (h_{33} * t_{37}))\ \%2$

New h_{30} =
$((h_{20} * t_{40}) + (h_{21} * t_{41}) + (h_{22} * t_{42}) + (h_{23} * t_{43}) + (h_{30} * t_{44}) + (h_{31} * t_{45}) + (h_{32} * t_{46}) + (h_{33} * t_{47}))\ \%2$

New h_{31} =
$((h_{20} * t_{50}) + (h_{21} * t_{51}) + (h_{22} * t_{52}) + (h_{23} * t_{53}) + (h_{30} * t_{54}) + (h_{31} * t_{55}) + (h_{32} * t_{56}) + (h_{33} * t_{57}))\ \%2$

New h_{32} =
$((h_{20} * t_{60}) + (h_{21} * t_{61}) + (h_{22} * t_{62}) + (h_{23} * t_{63}) + (h_{30} * t_{64}) + (h_{31} * t_{65}) + (h_{32} * t_{66}) + (h_{33} * t_{67}))\ \%2$

New h_{33} =
$((h_{20} * t_{70}) + (h_{21} * t_{71}) + (h_{22} * t_{72}) + (h_{23} * t_{73}) + (h_{30} * t_{74}) + (h_{31} * t_{75}) + (h_{32} * t_{76}) + (h_{33} * t_{77}))\ \%2$

NOTE: You don’t run the MixColum in the last round.

Key: The key schedule for AES is one of the most complex developed. We will not calculate round keys, but use the values given in the following table. The round key is XOR'd with the start state and the end state of each round.

Sample encryption for key = 6b5d and plaintext block = 2ca5

Round	Start	SubByte	ShiftRow	MixColumn	⊕	Round Key	=
input	2 a c 5				⊕	6 5 b d	=
1	4 f 7 8	8 d c 5	8 d 5 c	2 f 0 7	⊕	6 3 5 8	=
2	4 c 5 f	8 0 e d	8 0 d e	0 a e 3	⊕	1 2 e 6	=
3	1 8 0 5	4 5 a e	4 5 e a	d 9 2 8	⊕	7 5 d b	=
4	a c f 3	6 0 d b	6 0 b d		⊕	0 5 3 8	=
output	6 5 8 5						

The ciphertext is 6855.

To turn in, encrypt the plaintext 372c using the same key of 6b5d. This means you can use the round keys in the table above. You do not have to calculate them.

Lab 05 Template

1.) **How many times less secure is this (in theory vs in practice)?**
(10 points)

2.) **Functional code submitted to canvas**
(20 points)

3.) **What is k1 and k2?**
(10 points)

4.) **What is the secret message?**
(10 points)

5.) **Encrypt the plaintext 372c using the same key of 6b5d**
(30 points)

6.) **Functional code submitted to Canvas.**
(20 points)