1) Consider a cipher that uses a 16 bit key and 16 bit blocks. Let A and B both be permutations matrices used in the cipher, assuming that A and B are expressed in a similar manner to how IP is expressed in DES. Let C be a matrix that represents the equivalent permutation to applying A, followed by applying B. (Thus, C(x) = B(A(x)), where x is a 16 bit input.) Determine C given the matrices A and B below:

\[
A = \begin{bmatrix}
8 & 4 & 2 & 1 \\
12 & 6 & 5 & 3 \\
16 & 13 & 11 & 7 \\
15 & 14 & 10 & 9 \\
\end{bmatrix}
\quad B = \begin{bmatrix}
3 & 7 & 11 & 15 \\
2 & 6 & 10 & 14 \\
4 & 8 & 12 & 16 \\
1 & 5 & 9 & 13 \\
\end{bmatrix}
\]

**Solution**

1) \[
C = \begin{bmatrix}
2 & 5 & 11 & 10 \\
4 & 6 & 13 & 14 \\
1 & 3 & 7 & 9 \\
8 & 12 & 16 & 15 \\
\end{bmatrix}
\]

This follows a pretty simple formula. Essentially, \( B_1 = A_3 = 2 \), \( B_2 = A_7 = 5 \), and so on. All it requires is looking at the two matrices and keeping track of what indices the values are referring to. They go 1, 2, …, 16, rather than starting at 0.

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

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

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

**Solution**

2) \[
IP^{-1} = \begin{bmatrix}
15 & 5 & 9 & 13 \\
6 & 16 & 8 & 14 \\
4 & 1 & 11 & 10 \\
2 & 12 & 7 & 3 \\
\end{bmatrix}
\]

In this case, one is attempting to get the original configuration of the bits, in the order of 1, 2, 3, …, 16. So when looking at \( IP \), \( IP_{15} = 1 \), \( IP_3 = 2 \), and so on. This is how \( IP^{-1} \) is constructed.
3) If the input into all 8 S-boxes in DES is 1245789ABCDF, what is the output? Please express your output in 8 hexadecimal characters.

**Solution**

3) First, convert the input into bits. The input is 1245789ABCDF, or:

```
0001 0010 0100 0101 0111 1000 1001 1010 1011 1100 1101 1111
```

The next step is to separate the bits into eight six-bit blocks. I alternatively underlined and bolded these above just to make it easier to see. Each S-box takes one of these six-bit blocks and outputs four bits.

The S-boxes are included with the other DES tables on the website, underneath the section “Reference Sheets for Exams.” The AES tables are present here as well. In any case, values are pulled from the S-box as follows: the first and last bit of each six-block input are used to represent the row, and the middle four bits the column, of the desired output from the box. So for 000100, 00 means row 0, and 0010 means column 2. This will grab the value 13 or 1101 from S₁. Another intermediate step here is to convert each of these output values into four-bit blocks.

```
1101 0111 0101 0101 1101 0101 0101 0010
```

This can be represented in hexadecimal characters as **D755B552**.

4) 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 59E6BA91, what is the output of the expansion matrix. Express your answer as 12 hexadecimal characters.

**Solution**

4) First, convert the input once again into bits. The input is 59E6BA91, or:

```
0101 1001 1110 0110 1011 1010 1001 0001
```

Looking once again at the DES reference sheet shows the expansion permutation matrix E. It goes 32, 1, 2, 3, … etc. This essentially is just a manipulation of the above bits, with indices 1, 2, …, 32. Plugging the input into E results in:

```
101011 110011 111100 001101 010111 110101 010010 100010
```

And separating this into four-bit blocks and converting into hexadecimal results in an answer of **AF3F0D5F54A2**.
5) In the specification of DES, the key is represented as 64 bits, of which some are parity bits. Label all the bits (including parity bits) as \( k_1, k_2, \ldots, k_{64} \). If you knew the values of \( k_1 \) through \( k_{24} \), but had to perform a brute force search through the other bits of the key, how long, in the worst case, would it take you to find the key, given that you can search through \( 2^{20} \) keys in one second? Please express your answer in hours, minutes and seconds.

**Solution**
5) First, consider the data given by the question. The known bits are \( k_1, k_2, \ldots, k_{24} \), the parity bits are \( k_{32}, k_{40}, k_{48}, k_{56}, k_{64} \), and the unknown bits include everything else: \( k_{25}, \ldots, k_{31}, k_{33}, \ldots, k_{39}, k_{41}, \ldots, k_{47}, k_{49}, \ldots, k_{55}, k_{57}, \ldots, k_{63} \).

The key space is all possible permutations of the key, and to determine how long it will take to find the key in the worst case, you must first calculate the size of that key space. This is determined as follows:

\[
|\text{Key space}| = 2^{64} - 24 - 5 = 2^{35} \text{ keys}
\]

Seconds \( = (2^{35} \text{ keys}) \cdot \frac{1 \text{ sec}}{2^{20} \text{ keys}} = 2^{15} \text{ seconds} = 32768 \text{ seconds} \)

\( = 32768 \% 60 = 8 \text{ seconds} \)

Minutes \( = (32760 \text{ secs}) \cdot \frac{1 \text{ min}}{60 \text{ secs}} = 546 \text{ minutes} \)

\( = 546 \% 60 = 6 \text{ minutes} \)

Hours \( = (540 \text{ minutes}) \cdot \frac{1 \text{ hour}}{60 \text{ mins}} = 9 \text{ hours} \)

Thus, the answer is **9 hours, 6 minutes, and 8 seconds**.

6) Let the input to the MixCols (during AES encryption) be
\[
\begin{bmatrix}
A0 & 74 & 65 & 96 \\
2B & 8D & 2E & E3 \\
99 & 1F & C8 & 87 \\
C5 & E5 & F7 & BB
\end{bmatrix}
\]

What’s the output in row 3 col 4? (The matrix by which to “multiply” is
\[
\begin{bmatrix}
02 & 03 & 01 & 01 \\
01 & 02 & 03 & 01 \\
03 & 01 & 01 & 02
\end{bmatrix}
\]

**Solution**
6) To obtain the output for row 3, column 4, take the third row of the multiplication matrix and the fourth column of the input matrix. The calculation that needs to be performed is thus \( R_{3,4} = (01 \times 96) + (01 \times E3) + (02 \times 87) + (03 \times BB) \). Multiplication involves the techniques discussed in the lecture notes, and addition is simply applying XOR.

\[
\begin{align*}
01 \times 96 & = 96 \\
01 \times E3 & = E3 \\
02 \times 87 & = (87 \ll 1) \oplus 1B
\end{align*}
\]
\[ (1000 \ 0111 << 1) \oplus 1B \]
\[ = 1 \ 0000 \ 1110 \]
\[ \oplus \ 0001 \ 1011 \]
\[ = 15 \]

\[ 03 \times BB = (02 \times BB) \oplus (01 \times BB) \]
\[ = 6D \oplus BB \]
\[ = D6 \]

\[ 02 \times BB = (BB << 1) \oplus 1B \]
\[ = (1011 \ 1011 << 1) \oplus 1B \]
\[ = (1 \ 0111 \ 0110) \oplus 1B \]
\[ = 1 \ 0111 \ 0110 \]
\[ \oplus \ 0001 \ 1011 \]
\[ = 0110 \ 1101 \]
\[ = 6D \]

\[ 01 \times BB = BB \]
\[ 6D \oplus BB = 0110 \ 1101 \]
\[ \oplus 1011 \ 1011 \]
\[ = 1101 \ 0110 \]
\[ = D6 \]

\[ \text{Res}_{3,4} = (01 \times 96) + (01 \times E3) + (02 \times 87) + (03 \times BB) \]
\[ = 96 \oplus E3 \oplus 15 \oplus D6 \]
\[ = 1001 \ 0110 \]
\[ 1110 \ 0011 \]
\[ 0001 \ 0101 \]
\[ \oplus 1101 \ 0110 \]
\[ = 1011 \ 0110 \]
\[ = \text{B6} \]

Thus, the answer is \textbf{B6}.

As an aside, remember that when multiplying by 02 in AES, after performing the left shift on the input, the XOR with 1B is not necessary unless the original value of the input had a leftmost bit of 1. It just so happens that it was necessary both times in this problem.
7) In the key expansion algorithm of AES, if \( w[34] = \text{BB3A7920} \) and \( w[31] = \text{C659D034} \), what is \( w[35] \)?

**Solution**

7) In the key expansion algorithm of AES, if \( i \% 4 \neq 0 \), meaning \( w[i] \) is not a new round key, \( w[i] = w[i-4] \oplus w[i-1] \). \( 35 \% 4 = 3 \), so this is the only evaluation that needs to be done.

\[
\begin{align*}
w[31] &= \text{C659D034} \\
&= 1100 \ 0110 \ 0101 \ 1001 \ 1101 \ 0000 \ 0011 \ 0100 \\
w[34] &= \text{BB3A7920} \\
&= 1011 \ 1011 \ 0011 \ 1010 \ 0111 \ 1001 \ 0010 \ 0000 \\
x[35] &= w[31] \oplus w[34] \\
&= 1100 \ 0110 \ 0101 \ 1001 \ 1101 \ 0000 \ 0011 \ 0100 \\
&\oplus 1011 \ 1011 \ 0011 \ 1010 \ 0111 \ 1001 \ 0010 \ 0000 \\
&= 0111 \ 1101 \ 0110 \ 0011 \ 1010 \ 1001 \ 0001 \ 0100 \\
&= 7D63A914
\end{align*}
\]

8) Consider the process of AES Key Expansion. Imagine that we have:

\( w[28] = \text{B5 13 2F 76 (in hex)} \)
\( w[31] = \text{F4 9A 0D 8C (in hex)} \)

Calculate \( w[32] \), showing each of the following intermediate results: RotWord(temp), SubWord(RotWord(temp)), Rcon[i/4], and the result of the XOR with Rcon[i/4].

**Solution**

8) To calculate \( w[32] \), you must perform more intermediate steps because \( w[32] \) is a new round key.

RotWord(temp) involves the removal of the first byte of \( temp \) and appending it to the end. Temp is \( w[i-1] \), which in this case is \( \text{F4 9A 0D 8C} \). Thus, RotWord(temp) = \( \text{9A 0D 8C F4} \).

SubWord(RotWord(temp)) takes this output and then replaces each byte with values in the AES S-box. Given, for example, the first byte, 9A, the value is found at row 9, column A, which is B8. Thus, SubWord(9A 0D 8C F4) = B8 D7 64 BF.

Rcon[i/4] is Rcon[8], which is found in the Rcon table to be 80 00 00 00.

The XOR portion refers to performing an XOR between the SubWord output and the Rcon output.

\[
\begin{align*}
\text{B8D764BF} \oplus \text{80000000} &= 1011 \ 1000 \ 1101 \ 0111 \ 0110 \ 0100 \ 1011 \ 1111 \\
&\oplus 1000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \\
&0111 \ 1000 \ 1101 \ 0111 \ 0110 \ 0100 \ 1011 \ 1111
\end{align*}
\]
The final result involves then performing an XOR between the above value and \( w[i-4] \).
\[
\begin{align*}
38D764BF \oplus B5132F76 & = 0011\ 1000\ 1101\ 0111\ 0110\ 0100\ 1011\ 1111 \\
& \oplus 1011\ 0101\ 0001\ 0011\ 0010\ 1111\ 0111\ 0110 \\
& = 1000\ 1101\ 1100\ 0100\ 0100\ 1011\ 1100\ 1001 \\
& = 8D\ C4\ 4B\ C9
\end{align*}
\]
And now, the table is completely filled.

<table>
<thead>
<tr>
<th>RotWord</th>
<th>SubWord</th>
<th>Rcon[i/4]</th>
<th>XOR</th>
<th>FinalResult</th>
</tr>
</thead>
<tbody>
<tr>
<td>9A 0D 8C F4</td>
<td>B8 D7 64 BF</td>
<td>80 00 00 00</td>
<td>38 D7 64 BF</td>
<td>8D C4 4B C9</td>
</tr>
</tbody>
</table>

9) 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.

**Solution**

9) Each round key is 48 bits, and is essentially a rearrangement of the bits contained in the original 56-bit key. Eight bits, naturally, will be excluded. As there are 16 round keys, that means there are 16 x 48 bits used overall. However, \( \frac{16 \times 48}{56} = \frac{16 \times 6}{7} = \frac{96}{7} \). This is between 13 and 14, meaning that any given bit from the original 56-bit key cannot appear an equal number of times. Some will appear less, and others more. Bit 10, for example, appears slightly less often than average, as it appears only 13 times.

10) Using the code, AES.java, posted in the course examples, write a program (preferably in Java, but C, C++ and Python will be accepted as well), that prints out a 256 x 256 chart which has the results of every possible byte multiplication in the AES field. Your program should output 256 rows, where the \( i \)th row stores the products with byte value \( i-128 \). Your output should express each value as 2 hex chars followed by a space. (So, each row should have 256 x 3 = 768 characters on it, followed by a new line character.) The hex letters should be lower case and leading 0s should be printed. In Java, we can accomplish this as follows:

```java
System.out.printf("%02x ", byteToBePrinted);
```

Here is the beginning of the output of the few lines:

```
9b 03 82 28 a9 31 b0 7e ff 67
03 9e 1c bf 3d a0 22 fd 7f e2
82 1c 9f 3b b8 26 a5 75 f6 68
28 bf 3b 8a 0e 99 1d e0 64 f3
```
Solution
The key to solving this problem was reusing the method `mult(byte a, byte b)` from the posted AES code. This method called the `mult2` and `add` methods, so both of these need to be included as well. Then, in main, just run a double for loop in numeric order from -128 to 127. (I had messed this up in the description. For grading purposes, I'll accept starting at -127 also. Starting at -127 just puts the intended first row, for -128, last.) The code is in the included file, `multbytes.java`. 