

Dhruv Aggarwal  
HW2  
Aggarw45  
1/30

```

88 def encrypt(message,key,output):
89     file_in = open(key,"r")
90     key1 = BitVector(textstring = file_in.read())
91     key1 = key1.permute(key_permutation_1)
92     round_keys = generate_round_keys(key1)
93     bv = BitVector(filename= message)
94     output1 = open(output,"wb")
95     while (bv.more_to_read):
96         bitvec = bv.read_bits_from_file( 64 )
97         if bitvec.length() > 0:
98             if bitvec.length() < 64:
99                 bitvec = bitvec.pad_from_right(64-bitvec.length())
100             [LE, RE] = bitvec.divide_into_two()
101             for round_key in round_keys:
102                 newRE = RE.permute( expansion_permutation )
103                 out_xor = newRE^( round_key )
104                 re_mod = substitute(out_xor)
105                 re_mod = re_mod.permute(permutation_box)
106                 temp = RE
107                 RE = LE ^ re_mod
108                 LE = temp
109             final_string = RE + LE
110             final_string.write_to_file(output1)
111
112 def decrypt(message,key,output):
113     file_in = open(key)
114     key1 = BitVector(textstring= file_in.read())
115     key1 = key1.permute(key_permutation_1)
116     round_keys = generate_round_keys(key1)
117     round_keys = round_keys[::-1]
118     bv = BitVector(filename= message)
119     output1 = open(output,"wb")
120     while (bv.more_to_read):
121         bitvec = bv.read_bits_from_file( 64 )
122         if bitvec.length() > 0:
123             if bitvec.length() < 64:
124                 bitvec = bitvec.pad_from_right(64-bitvec.length())
125             [LE, RE] = bitvec.divide_into_two()
126             for round_key in round_keys:
127                 newRE = RE.permute( expansion_permutation )
128                 out_xor = newRE^( round_key )
129                 re_mod = substitute(out_xor)
130                 re_mod = re_mod.permute(permutation_box)
131                 temp = RE
132                 RE = LE ^ re_mod
133                 LE = temp
134             final_string = RE + LE
135             final_string.write_to_file(output1)
136
137 if sys.argv[1] == "-e":
138     encrypt(sys.argv[2],sys.argv[3],sys.argv[4])
139
140 elif sys.argv[1] == "-d":
141     decrypt(sys.argv[2],sys.argv[3],sys.argv[4])
142

```

Encrypted output:

[illegible]

Decrypted Output:

Earlier this week, security researchers took note of a series of changes Linux and Windows developers began rolling out in beta updates to address a critical security flaw: A bug in Intel chips allows low-privilege processes to access memory in the computer's kernel, the machine's most privileged inner sanctum. Theoretical attacks that exploit that bug, based on quirks in features Intel has implemented for faster processing, could allow malicious software to spy deeply into other processes and data on the target computer or smartphone. And on multi-user machines, like the servers run by Google Cloud Services or Amazon Web Services, they could even allow hackers to break out of one user's process, and instead snoop on other processes running on the same shared server. On Wednesday evening, a large team of researchers at Google's Project Zero, universities including the Graz University of Technology, the University of Pennsylvania, the University of Adelaide in Australia, and security companies including Cyberus and Rambus together released the full details of two attacks based on that flaw, which they call Meltdown and Spectre.

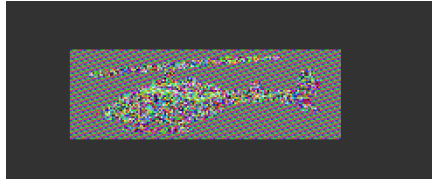
DES\_image.py:

```

80 def encrypt(message,key,output):
81     file_in = open(key,"r")
82     key1 = bytearray(teststring + file_in.read())
83     key1 = key1.permutate(key.permutation)
84     round_keys = generate_round_keys(key1)
85     final_output = open(output,"wb")
86     test_file = open("test.ppm","r")
87     with open(message,"rb") as whole:
88         entire_block.readlines()[:8]
89         final_output.write(entire_block[:8])
90     with open(message,"rb") as whole:
91         non_header = whole.readlines()[:3]:
92         test_file.write(non_header)
93         bv = bytearray(message.read("test.ppm"))
94         while (bv.more_to_read):
95             bitvec = bv.read_bits_from_file(64)
96             if bitvec.length() > 8:
97                 if bitvec.length() < 64:
98                     bitvec.pad_from_right(64-bitvec.length())
99                 LE = bitvec.data_into_ints_two()
100                 for round_key in round_keys:
101                     newRE = RE.permutate( expansion_permutation(
102                         xor_xor = newRE ^ round_key )
103                     )
104                     re_mod = substitute_output_xor)
105                     test_file.write(re_mod.permutate(permutation_box))
106                     temp = RE
107                     RE = LE ^ re_mod
108                     LE = temp
109                 final_string = RE + LE
110                 final_string.write_to_file(final_output)

```

Decrypted Image:



Explanation:

The way I did it was by using the professors script and following his notes. I followed the steps of DES. I basically encrypted in chunks of 64 bits. Split the 64 bits into two halves and then did expansion permutation and substitution and then expanded it again. Added the two halves back. For decryption I used the same code as encryption except I reversed the round keys. For the image I read the header bits and added it to the final image. Then took the rest of it and made a temp.ppm. Then I followed the same steps as encryption of the text except with the temp.ppm and added that to the final image. I know in the professor's pdf it says that you shouldn't have to use the same method and that there is an easier one but I could not figure that out.