The python program DES.py implements the full DES algorithm. Given an encryption key and some plaintext, DES.py will write a file containing the correct encryption and decryption result. When command-line below is invoked, DES.py will perform DES encryption on the plaintext in message.txt using the key in key.txt and write the ciphertext to a file called encrypted.txt. '-e' symbolizes that DES.py will run the encryption process.

```
python DES.py -e message.txt key.txt encrypted.txt
```

Figure I. Example of Command-Line Syntax to Invoke DES Encryption using DES.py

An example of this is when a message.txt contains the following text:

Scuderia Ferrari is the racing division of luxury Italian auto manufacturer Ferrari and the racing team that competes in Formula One racing. The team is also known by the nickname "The Prancing Horse", in reference to their logo. It is the oldest surviving and most successful Formula One team, having competed in every world championship since the 1950 Formula One season. The team was founded by Enzo Ferrari, initially to race cars produced by Alfa Romeo. By 1947 Ferrari had begun building its own cars. Among its important achievements outside Formula One are winning the World Sportscar Championship, 24 Hours of Le Mans, 24 Hours of Spa, 24 Hours of Daytona, 12 Hours of Sebring, Bathurst 12 Hour, races for Grand tourer cars and racing on road courses of the Targa Florio, the Mille Miglia and the Carrera Panamericana. The team is also known for its passionate support base, known as the tifosi. The Italian Grand Prix at Monza is regarded as the team's home race.

Figure II. Contents of the used message.txt

When the above command-line is invoked, DES.py will output the following to encrypted.txt:

0c46d7cd5b7efc319691493448bb36733af8d5e4da962e15e85db329c5031857a154f62cbfb7c82d298c9456ef29adb
8e86cc51ae7f025097f513677406336598e0f3f1f0c5ecaf0b55649222b19a27da886fa8c4d2b9e0e88a2745b99e6bbb
4658cd9fd3606e05d11919eddd39723e333aa813ebd9a9ae6810271c9d634cba829e1b7a82bd994073d054e62a79d
8bbd1ebe00d2288b8c05b0f4d5ec799e3f7d5db8b04a23106d0151c6fea8bd1826a92e611e73a1bc4949ed703d017
4516196ef7faed8a411c7efc9b11b6b44fa864c7692c80a7ac2dc6f5d467e8b6588845f5c8c1f4493c9d94f3af8d5e4da
962e1580d4d42e93e281c6aab31eec856fead76a96c9d84c4a3fce61ded79fdd9a943cb446a58d881c211b5ba21a1dc
81659123283460d36ca20cba580ebd51188824724ec416aebeff0d01d2be942433af7679b2d5d55a4b8c931151283e
60d8e99e90701d26b28a139a46c209a2a93f6250b902ff25ee8aa0f56ea075b13c3ca4dbd985da7338582b48b412c3
3ce01dc4bcbb7cb9a3e905deb0caf473c5b801aa2872c62d06d015b9b7aba88a48889f7b2cd6602ec4311480ef124ad
ff91a834630b41c2f4d29769ca093ec31ee4779264af3a6ecd51cc098d3acfb1c5fdeff53a694ea26c872220eb2c75894
e9e10b1beba091a61279d20154b4c46eda9c3d6b6df07eaaa1dc93f98246eefeb34d8ea72bef7558055080ed4d73afe
523bb6723e79ba8eae813579fc2f74a2a64cdf2484bc8267b7c0b0cc28ab5ba21a1dc8165912c99d911d997a8e8298
53c23bcd8681544a3bc6ea2a56ae5844873d757d272114000874af4a2adff08a824e0c1b8dbbb72a02f86fb4c95668b
5bdcb5c3c3d3fc3545d14e6459f7d2b7050edc71e4c58ad593b284e6fee59f41bf13fddf342694530d4e70c288d9a61e
3515a37674fbb7bc98730a9d700b5c8d332cc75c1a41e39a2ae33cb95d43e92b3f168a97488f8a7cfbe9993019259ed
8cfdc1cddb6e60cb40803c3e931e1278d85ae80815e10b3a7496e30b24e6b996e2400cad3f3999fdab7d3bcf897a9a3
76e85932b9d711e634dcf3a756b2a93165df4a192bf0d0a271415986d5e1dbd019250095819c5e0b55b095bbb94a0
0a009e6c9e6a998598c2f98075a8861a43710dbd6cb63a94d66c2d4d779ead4200ef8f58a2d2c3ab25ccd2fec9c8489
ab4b8bb1c95b3b7da5d9b5eb50e9733bdf981112601bec9feb807ef32f154f825a870d7ff1ec081545d343c085bb0bc
7b2bee895410488ad30eaec469d6170b2a502a616b4b55e49e7ab3517db4259cc90e91b70e232ec1f8a1ea85a1b4d
4c63fa94fc1b80e7005183f54ace18926dbf3330252ca26895d60dd71

Figure III. Contents of the written output of to encrypted.txt using DES Encryption in DES.py

It is important to note that the output file of the DES encryption process of DES.py is saved as a single-line hex file. The overall DES encryption process is in accordance with the standard of Data Encryption Algorithm, where it first will generate the 56-bit encryption key using the provided key and then generate the round keys from the encryption key. The algorithm would then continuously read 64-bit block from

the file to be encrypted and would pad enough bits to ensure that the length is 64. Each of the 64-block will go through the following:

- The left half and right half will be switched so that the new left half is the previous right half of the 64-bit input section
- The new right half is then computed as follows:
  - The previous right half is split into 4-bit segments and is then permuted into 6-bit segments using the last bit of its byte predecessor and first bit its byte successor, respectively. This generates a 48-bit result.
  - The result is XORed with the round key.
  - Substitution is performed on the XOR result in accordance with predefined S-boxes and will generate a 32-bit result.
  - Another permutation is performed using predefined P-box.
  - The result of the P-box permutation is then XORed with the previous left half.
  - The XOR result is then assigned as the new right half of the 64-bit block.

For further reference, below is the DES encryption algorithm implemented in DES.py:

```
def encrypt ( self , message_file , outfile ):
    # encrypts the contents of the message file and writes the ciphertext to the outfile
    bv = BitVector(filename=message_file)
    output_file = open(outfile, 'w')
    while bv.more_to_read:
        bitvec = bv.read_bits_from_file(64) # process 8 bytes
        if bitvec.length() % 64 != 0:
            bitvec.pad_from_right(64 - bitvec.length() % 64)
        [LE, RE] = bitvec.divide_into_two()
        if bitvec.length() > 0:
            for round_key in self.round_keys:
                newLE = RE.deep_copy()
                RE = RE.permute(self.expansion_permutation) # permute 32 bits to 48 bits
                out_xor = RE ^ round_key # XOR with round key
                s_output = self.substitute(out_xor) # substitute with S boxes
                p_output = s_output.permute(self.p_box_permutation)  # permute with P box
                RE = LE ^ p_output
                LE = newLE
        bitvec = RE + LE
        output_file.write(bitvec.get_bitvector_in_hex())
    output_file.close()
```

Figure IV. DES Encryption algorithm of DES.py

The process of generating the 56-bit encryption key and round keys is done in the __init__ method of the DES class (class constructor when creating a DES object), which is the following:

```
def __init__ ( self , key ):
    with open(key) as key_file:
        key_text = key_file.read()

    encryption_key = self.get_encryption_key(key_text) # generate 56-bit encryption key
    self.round_keys = self.generate_round_keys(encryption_key) # generate round keys
```

Figure V. __init__ method of the DES class

Furthermore, to invoke a DES decryption on such file called encrypted.txt using a key in key.txt with the output written to decrypted.txt, the following command-line is used, where '-d' symbolizes the usage of DES decryption:

```
python DES.py -d encrypted.txt key.txt decrypted.txt
```

Figure VI. Example of Command-Line Syntax to Invoke DES Decryption using DES.py

It might be important to note that the algorithm used for the DES encryption and DES encryption are very similar, in that it only differs the direction of XOR with round key is the reverse direction for DES decryption. Below is the algorithm used for the DES decryption algorithm in DES.py:

```
def decrypt ( self , encrypted_file , outfile ):
    # decrypts contents of encrypted_file and writes the recovered plaintext to the outfile
    with open(encrypted_file, 'r') as file:
        bv = BitVector(hexstring = file.read())
    output_file = open(outfile, 'w')
    num_iter = round(bv.length()/64)
    for x in range(num_iter):
        bitvec = bv[(x*64):(x*64)+64]
        [LE, RE] = bitvec.divide_into_two()
        if bitvec.length() > 0:
            for round_key in self.round_keys[::-1]: # reverse the direction of round keys
                newLE = RE.deep_copy()
                RE = RE.permute(self.expansion_permutation)
                out_xor = RE ^ round_key
                s_output = self.substitute(out_xor)
                p_output = s_output.permute(self.p_box_permutation)
                RE = LE ^ p_output
                LE = newLE
        bitvec = RE + LE
        output_file.write(bitvec.get_bitvector_in_ascii())
    output_file.close()
```

Figure VII. DES Decryption algorithm of DES.py

An example of the DES decryption algorithm can be performed using a file called encrypted.txt that contains the cipher text in Figure II (the output file of DES encryption), which, when invoked using the command-line in figure VI, DES.py writes the following into decrypted.txt:

Scuderia Ferrari is the racing division of luxury Italian auto manufacturer Ferrari and the racing team that competes in Formula One racing. The team is also known by the nickname "The Prancing Horse", in reference to their logo. It is the oldest surviving and most successful Formula One team, having competed in every world championship since the 1950 Formula One season. The team was founded by Enzo Ferrari, initially to race cars produced by Alfa Romeo. By 1947 Ferrari had begun building its own cars. Among its important achievements outside Formula One are winning the World Sportscar Championship, 24 Hours of Le Mans, 24 Hours of Spa, 24 Hours of Daytona, 12 Hours of Sebring, Bathurst 12 Hour, races for Grand tourer cars and racing on road courses of the Targa Florio, the Mille Miglia and the Carrera Panamericana. The team is also known for its passionate support base, known as the tifosi. The Italian Grand Prix at Monza is regarded as the team's home race.

Figure VIII. Contents of the written output to decrypted.txt using DES Decryption in DES.py

Note that the extra spaces at the end are caused by padding by zeroes (null byte characters) when the plaintext bit size is not divisible by the DES block size.

DES.py is also capable of performing DES encryption on an image. More specifically, given an image in PPM format, DES.py will perform DES encryption only on the image data and will not encrypt the PPM header. Using the following command line below, DES.py will perform DES encryption on the image in image.ppm with key in key.txt and store the encrypted image in image enc.ppm, additionally, the '-i' in the command-line syntax symbolizes the usage of DES.py to perform DES encryption on ppm image file:

```
python DES.py -i image.ppm key.txt image_enc.ppm
```

Figure IX. Example of Command-Line Syntax to Invoke DES Encryption on an Image File in PPM Format using DES.py

The following method to handle image file is added to DES class in DES.py to account for header file that are present in files with PPM format.

```
def encrypt_img( self , input_file , outfile ):
    with open(input_file, 'rb') as file1:
        # read the first three lines of the PPM header
        header_length = file1.readline()
        header_length2 = file1.readline()
        header_length3 = file1.readline()
    bv = BitVector(filename=input_file)
    output_file = open(outfile, 'wb')
    # write the header to the first three lines of the output file
    output_file.write(header_length)
    output_file.write(header_length2)
    output_file.write(header_length3)
    while bv.more_to_read:
        bitvec = bv.read_bits_from_file(64)
        if bitvec.length() % 64 != 0:
            bitvec.pad_from_right(64 - bitvec.length() % 64)
        [LE, RE] = bitvec.divide_into_two()
        if bitvec.length() > 0:
            for round_key in self.round_keys:
                newLE = RE.deep_copy()
                RE = RE.permute(self.expansion_permutation)
                out_xor = RE ^ round_key
                s_output = self.substitute(out_xor)
                p_output = s_output.permute(self.p_box_permutation)
                RE = LE ^ p_output
                LE = newLE
        bitvec = RE + LE
        bitvec.write_to_file(output_file)
    output_file.close()
```

Figure X. DES Encryption algorithm for PPM files of DES.py

It is important to note that the DES encryption algorithm for PPM files is the same as DES encryption algorithm mentioned in Figure IV, although there are a few additions that need to be added to account for the PPM header. The DES encryption algorithm for PPM file will also not write a single-line hex string to the output file, instead it will write the encrypted image data directly to the output file. This is done so that the output will be viewable in PPM format. An example of DES encryption for PPM files can be performed by invoking the command-line illustrated in Figure IX and using a file called image.ppm that displays the following image:

Figure XI. Image contained in image.ppm

The result of the DES encryption is the following image:

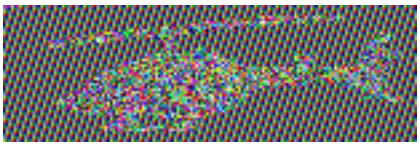

Figure XII. DES Encrypted Image Output contained in image_enc.ppm

For reference, below is any of the user-defined functions used in DES.py but not mentioned in the document:

```
if __name__ == '__main__':

    option = sys.argv[1]
    input_file = sys.argv[2]
    key_file = sys.argv[3]
    outfile = sys.argv[4]

    cipher = DES(key=key_file)
    if option == '-e':
        cipher.encrypt(input_file, outfile)
    elif option == '-d':
        cipher.decrypt(input_file, outfile)
    elif option == '-i':
        cipher.encrypt_img(input_file, outfile)
```

Figure XIII. __main__ function of DES.py

```
def get_encryption_key(self, key_string):
    key = BitVector(textstring=key_string)
    key = key.permute(self.key_permutation_1)
    return key
```

Figure XIV. Function to Generate Encryption Key

```
def generate_round_keys(self, encryption_key):
    round_keys = []
    key = encryption_key.deep_copy()
    for round_count in range(16):
        [LeftKey, RightKey] = key.divide_into_two()
        shift = self.shifts_for_round_key_gen[round_count]
        LeftKey << shift
        RightKey << shift
        key = LeftKey + RightKey
        round_key = key.permute(self.key_permutation_2)
        round_keys.append(round_key)
    return round_keys
```

Figure XV. Function to Generate Round Keys

```
def substitute(self, expand_half_block):
    output = BitVector(size=32)
    segment = [expand_half_block[x * 6:x * 6 + 6] for x in range(8)]
    for index in range(len(segment)):
        row = 2 * segment[index][0] + segment[index][-1]
        column = int(segment[index][1:-1])
        output[index * 4:index * 4 + 4] = BitVector(intVal=self.s_boxes[index][row][column], size=4)
    return output
```

Figure XVI. Function to Generates the 32-bit S-box Substituted Result