1. (text 3.5) Suppose the DES mangler function maps every 32-bit value to zero, regardless of the value of its input. What function would DES then compute?

DES does the following (see text figure 3-2):
- Initial permutation
- 16 DES rounds
- Swap left and right halves
- final permutation (inverse of initial permutation)

With a mangler function that outputs 0 always, each DES round just swaps L and R. So after 16 (even number) DES rounds, the initial 64-bit word would be unchanged. So DES would do the following:
- Initial permutation
- Swap left and right halves
- final permutation

Based on the initial permutation, the net result is a permutation that interchanges consecutive even and odd bits.
[If the swap were not there, DES would have no affect at all.]

Grading key [out of 5 points]:
1 point for just writing something.
2 points for saying that each DES round just exchanges L and R.
3 points for saying that each DES round just exchanges L and R, so after 16 (even) rounds, there is no change.
4 points: if you miss the final L-R swap and just say that DES has no effect.
5 points: if you get the answer.

2. (text 3.8) Why is a DES weak key its own inverse? (Hint: DES encryption and decryption are the same once the per-round keys are generated.)

For a DES weak key, each of C₀ and D₀ is equal to all ones or all zeros. Each Cᵢ is a permutation of C₀, so each Cᵢ equals C₀. Each Dᵢ is a permutation of D₀, so each Dᵢ equals D₀. Kᵢ depends only on Cᵢ and Dᵢ, so all Kᵢ’s are equal. So the sequence K₁, K₂, ⋯, K₁₆ is the same as the sequence K₁₆, K₁₅, ⋯, K₁. So the encryption operation is the same as the decryption operation (because decryption is the same as encryption but with the keys in reverse order).

Explanation of the hint: In case the hint is not clear, here are more details (from slides):

```
DES_encryption {
    initial permutation to get L₀|R₀ from data block
    for n=1, 2, ⋯, 16 do L₀|R₀ := Eₙ(Kₙ, Lₙ⁻¹|Rₙ⁻¹),
        where Eₙ denotes the computation of encryption round n.
        swap left and right halves, yielding R₁₆|L₁₆
        inverse of initial permutation, yielding cipher block
}
```
DES_decryption {
    initial permutation of cipher block, yielding R_{16}|L_{16}
    for n = 16, 15, ..., 1 do R_{n-1}|L_{n-1} := D_n(K_n, R_n|L_n),
        where D_n denotes the computation of decryption round n.
    swap left and right halves, yielding L_0|R_0
    inverse of initial permutation, yielding data block.
}

Section 3.3.4 explains that decryption round n is identical to encryption round n with L_n and R_n swapped, i.e., D_n(K_n, R_n|L_n) equals E_n(K_n, L_n|R_n). Substituting this in DES_decryption, we see that the only difference between encryption and decryption is that the K_n’s are used in the opposite order. So there is no difference if all the K_i’s are the same.

Grading Key [Out of 5 points]
1 Points for mentioning what are the properties of weak keys.
2 Points for explaining the generation of per round keys for DES.
2 Points for explaining what would happen if the sequence K_1, K_2, ..., K_{16} is the same as the sequence K_{16}, K_{15}, ..., K_1.
1 Point for getting the answer.

3. (text 4.1) What pseudo-random block stream is generated by 64-bit OFB with a weak DES key.

The OFB pad sequence is E_x(IV), E_x(E_x(IV)), E_x(E_x(E_x(IV))), ...
A weak key is its own inverse, i.e., for any block b: E_x(b) = D_x(b). So E_x(E_x(b)) = b.
So the resulting OFB pad sequence is E_x(IV), IV, E_x(IV), IV, ....

Grading Key [Out of 5 points]
2 Points for showing the correct OFB pad sequence with a weak DES key.
2 Points for showing that E_x(b) = D_x(b). So E_x(E_x(b)) = b for a weak key as it is own inverse.
1 Point for showing the resulting OFB pad sequence.

4. (text 4.2) The pseudo-random stream of blocks generated by 64-bit OFB (i.e., K{IV}, K{K{IV}}, ...) must eventually repeat. Will K{IV} necessarily be the first block to be repeated. Explain.

IV will be the first block to repeat.
Proof:
For brevity, let b_i denote the i-fold encryption of IV.
So the pad sequence is b_1, b_2, b_3, ...., where b_{i+1} is the encryption of b_i and b_i is the decryption of b_{i+1} (because decryption is the inverse of encryption).
Let b_k be the first repeat element and let b_k = b_j where j < k.
• If j=1 we are done.
• If j > 1 then b_{j-1} = b_{k-1} (since b_j = b_k). So b_k is not the first repeat element. Contradiction.
So b_k = b_1.
Note that we only needed the fact that encryption is reversible.
5. (text 5.1) Would it be reasonable to compute an RSA signature on a long message m by signing \( m \mod n \) (i.e., using \((m \mod n)^d \mod n\) as the signature).

No. Recall that RSA restricts the message to be signed to be smaller than n.

If m is larger than n, then message m and message \((m \mod n)\) would have the same signature. So it would be easy to generate different messages that have the same signature.

6. (text 5.6) Why do MD4, MD5, and SHA-1 require padding of messages that are already a multiple of 512-bits?

Otherwise it would be easy to find two messages with the same hash. Let M' be any message that is not a multiple of 512 bits. Let M be M' padded as in MD4, so M is a multiple of 512 bits. If no padding is used for M (because it is a multiple of 512 bits) then MD4(M) would be the same as MD4(M').

7. (text 6.8) Given your RSA signature on \( m_1 \) and \( m_2 \), how can one compute your signature on \( m_1^j \cdot m_2^k \) for any positive integers j and k.

Let \( s_1 \) be the signature of \( m_1 \), i.e., \( s_1 = m_1^d \mod n \).
Let \( s_2 \) be the signature of \( m_2 \), i.e., \( s_2 = m_2^d \mod n \).
Signature(\( m_1^j \)) = \( s_1^j \mod n \) [because \( (m_1^j)^d \mod n = (m_1^d)^j \mod n \)].
Signature(\( m_1 \cdot m_2 \)) = \( s_1 \cdot s_2 \mod n \) [because \( (m_1 \cdot m_2)^d \mod n = (m_1^d \cdot m_2^d) \mod n \)].
Signature(\( m_1^j \cdot m_2^k \)) = \( s_1^j \cdot s_2^k \mod n \) [from above].

Grading key [out of 5 points]:
1 Point to show what a signature on \( m_1 \) looks like.
2 Points to show what would Signature(\( m_1^j \)) be in terms of \( m_1^d \).
2 Points to get the final computation i.e. Signature(\( m_1^j \cdot m_2^k \)) = \( s_1^j \cdot s_2^k \mod n \).
8. Using the efficient algorithm, compute $131^{25} \mod 15$.

$25 = (11001)_2 \quad [25 = 16 + 8 + 1]$

$131^{(1)} \mod 15 = 11$

$131^{(10)} \mod 15 = 11 \cdot 11 \mod 15 = 121 \mod 15 = 1$

$131^{(11)} \mod 15 = 1 \cdot 11 \mod 15 = 11 \mod 15 = 11$

$131^{(110)} \mod 15 = 11 \cdot 11 \mod 15 = 121 \mod 15 = 1$

$131^{(1100)} \mod 15 = 1 \cdot 1 \mod 15 = 1$

$131^{(11000)} \mod 15 = 1 \cdot 1 \mod 15 = 1$

$131^{(11001)} \mod 15 = 1 \cdot 11 \mod 15 = 11$

So $131^{25} \mod 15 = 11$

9. Suppose a plaintext file of 5 MB is encrypted with a secret-key algorithm (e.g., DES, AES), and the resulting file is compressed with a lossless compression algorithm (e.g., zip), and the resulting file is 3 MB. What does this imply about the plaintext, about the encryption algorithm, and about the compression algorithm.

It implies that the encryption algorithm is weak. A good algorithm would randomize the plaintext sufficiently so that the compression algorithm would not be able to find structure to exploit.

It does not imply anything about the plaintext or the compression algorithm.