Eric Rothman
Practical Cryptogaphic Systems
Writing HW 1

Problem 1:

1) What is the key-space for a four-rotor navy enigma?

With the four rotors in place, the initial key size for setting the rotors was 26^4. While it may originally seem that adding a fourth rotor would make the possible ordering of the wheels go from 3! to 4!, the fourth rotor could not be exchanged with the other three, so it didn't change that facet of the key size. Finally adding the plug board that all navy enigmas had increased the key size by 26!, if everything could be wired. For the actual enigma machines, there existed only 10 wires so the plug board only increased the key size by 150,738,274,937,250 which was arrived at using the formula $26! / ((26 - 2n)! \cdot n! \cdot 2^n)$ for n number of cables.
This also ignores the fact though that there were multiple routers that could be placed into the machine. So ignoring the routers we get $150,738,274,937,25 * 26^4 * 6 = 4.133*10^{17}$.

If you include the ability to change routers, of which there were 8 for the first 3 and 2 for the fouth one, and including the ability to reflect the pairs, there are $3.1*10^{25}$ keys.

Finally if you just base the answer off the Handbook of Applied Cryptography which says adding the plug board increased the key size by 26! and you didn't know that the fourth rotor couldn't be exchanged, the key size is $4.424*10^{33}$.

I used this source for my initial 2 answers. http://users.telenet.be/d.rijmenants/en/enigmatech.htm

2) The Lucky13 attack relies on changing the size of the input to HMAC so that it crosses a 64-byte boundary. Why?

The Lucky13 attack needs to trick the HMAC decoder that there is padding in the encrypted output. If the decode thinks there is padding it will remove some bytes from the plain text to verify MAC. What the Lucky13 attack does is that it makes it so that the amount of padding removed is actually based off of plain text, either raw plain text or plain text xor with a sequence the attack created. In that case the message will fail the MAC test and an error message is returned. The timing on when the message is returned is based off of how many padding bytes were removed. The number of padding bytes removed is decided by the last bytes of the message, which the attack makes to be plain text. So using this knowledge and the knowledge of how long it took for an error message to be returned, the attacker can figure out the last bytes of the last block of the encrypted plain text. This can then be repeated until the entire block is decrypted.

3) Describe the complementation property of the DES cipher

The complementation property of the DES cipher is that if you bitwise complement both the key and the plain text, the result after encrypting would be the bitwise complement of the result when you run the key and plain text through normally.

So in other words, “complimenting both the key and the plain text results in complimenting the DES cipher”.

4) what is the danger of using a “two-time pad”. More concretely, what can happen if I produce two equal-length ciphertexts $C_1 = M_1$ xor K and $C_2 = M_2$ xor K where both M are different messages but K is the same key. Why is this bad?

This is bad for the example give because if you xor both cipher texts, the K will cancel out leaving you with the xor of the two plain text messages. So if you know one of the values of M, then you can figure out the other one by xor the M you know with the two cipher texts.

If the two cipher test have redundancy in them, which would occur a lot if they were both messages of text, the redundancies would cancel out allowing you to figure out things about the message and the key used to encrypt it. With enough redundancies, the original messages can be recovered.

5) The KRACK paper relied on forcing nonce re-use for an encryption scheme. Explain the possible implications of this for security. How does this relate to your previous answer?

If you can force nonce re-use than it is possible to replay, decrypt, and forge packets in a specific communication direction. So if nonce re-use is forced it becomes possible to decrypt all messages transmitted, so no message is secure anymore. This is very similar to what happens in the previous answer. Nonce are only supposed to be used once, so they are equivalent to a one time pad. The moment that it used two time it becomes possible to use that information to decrypt the messages transmitted.

Problem 2:

So we know that $SuperDESEnc(K_1||K_2, M) = DESEnc_{K2}(DESEnc_{k1}(M))$.
Let $C = SuperDESEnc(K_1||K_2, M)$, so $C = DESEnc_{K2}(DESEnc_{k1}(M))$.
So $DESDec_{K2}(C) = DesDec_{k2}(DESEnc_{k2}(DESEnc_{k1}(M)))$, which implies $DESDec_{K2}(C)=DESEnc_{k1}(M)$.

This idea leads to the strategy of the attack.

First the attacker generates a message M_1 for the encryptor to encrypt though $SuperDESEnc(K_1||K_2, M_1)$ to get a cipher text C_1.

Next the attacker goes through every value k_i for K_1 and generates $E_i = DESEnc_{ki}(M)$ for each value. Then the attacker goes through every value k_j for K_2 and generates $D_j = DESDec_{kj}(C_1)$. After every E_i and D_j is calculated, the attacker goes through each value and when $E_i = D_j$, they note that k_i and k_j are a potential pair.

When the attacker has calculated every potential pair, they generate a new message M_2 that is unique from M_1 and send it to the encyptor to get $C_{2.}$.

The attacker then tries every pair of k_i and k_j to try and decrypt C_2 to get M_2.

If multiple pairs decrypt successfully, the attacker repeats generating unique messages M_a until only one pair decrypts them all successfully.

2^{56} encryptions are necessary to generate all values of E_i.

Likewise 2^{56} decryptions are necessary to generate all values of D_j.

The number of decryptions necessary to remove all candidates cannot be calculated until the strategy actually activates, but if M_2 is carefully chosen for which only one potential pair decrypts it, then there are decryptions equal to the number of candidate pairs, which is less than $2^{56.}$. Lets call the number of decryptions to remove all candidates A.

So there are $2^{56} + 2^{56} + A = 2^{57} + A < 2^{58}$ encryptions/decryptions to figure out the values for the two keys.

The amount of storage necessary is 2^{57} messages of length $|M_1|$.