Modern Cryptography
600.442
Homework 1 Solutions

Dr. Christopher Pappacena

Due September 12, 2013

Problem 1.5: The point here is that, given a known plaintext and corresponding ciphertext, we can look up the letters of the substitution. In order to recover the entire key, we need to see each of the letters A - Z at least once. Of course, we may not ever see a rare letter like Q or Z, and as long as it does not show up later in the ciphertext, we don't ever need to see it. So a better answer is that once we have seen every letter which occurs in the plaintext, we have all the information we need to decrypt the rest of the ciphertext.

For example, if we know that the plaintext is THE CAT IN THE HAT and we see the ciphertext CBR QZC PV CBR BZC, we know how to decrypt subsequent occurrences of the letters A, C, E, H, I, N, and T.

Problem 1.6: For a chosen plaintext attack, things are even easier. All the adversary needs to do is ask for the encryption of the 26-letter message ABCDEFGHIJKLMNOPQRSTUVWXYZ. This gives him the entire secret key!

Problem 2.2: The statement is false. In fact, for a perfectly secret encryption scheme we have

$$\Pr[M = m|C = c] = \Pr[M = m'|C = c] \Longleftrightarrow \Pr[M = m] = \Pr[M = m'] \quad (1)$$

So for any nonuniform distribution on $\mathcal{M}$ we can find m, m' which refute the statement.

To prove equation (1) we rewrite $\Pr[M = m|C = c]$ using Bayes's Theorem:

$$\Pr[M = m|C = c] = \frac{\Pr[C = c|M = m] \times \Pr[M = m]}{\Pr[C = c]}$$

On the one hand, if

$$\Pr[M = m|C = c] = \Pr[M = m'|C = c],$$

then applying Bayes's Theorem to both sides and canceling the common denominator of $\Pr[C = c]$ gives

$$\Pr[C = c|M = m] \times \Pr[M = m] = \Pr[C = c|M = m'] \times \Pr[M = m'].$$

Since Π is perfectly secret, $\Pr[C = c|M = m] = \Pr[C = c|M = m']$. Canceling this common term gives

$$\Pr[M = m] = \Pr[M = m'].$$

This process is reversible: Starting with $\Pr[M = m] = \Pr[M = m']$ we can multiply through by $\Pr[C = c|M = m]$ or $\Pr[C = c|M = m']$, divide by $\Pr[C = c]$, and use Bayes's Theorem to conclude that

$$\Pr[M = m|C = c] = \Pr[M = m'|C = c].$$

Problem 2.3: The one-time pad with nonzero keys is *not* perfectly secret, by Theorem 2.7. Specifically, $|\mathcal{K}| = 2^n - 1$ is less than $|\mathcal{M}| = 2^n$, so by Theorem 2.7 it cannot be perfectly secret.

To reconcile this with the fact that the 0 key does not change the plaintext, we note that $k = 0$ is no more or less likely to be selected than any other key. Suppose an adversary $\mathcal{A}$ prepares two messages m_0 and m_1 and is returned m_0. There are two ways that this can happen – either the the message m_0 was encrypted with $k = 0$ or the message m_1 was encrypted with $k = m_0 \oplus m_1$. Either key is equally likely so that, even in the case where one of the messages is returned as a ciphertext, $\mathcal{A}$ is still no more or less likely to guess the correct message than at random.

Problem 2.5: This is false because there is no specification of how the key is used to encrypt. For a ridiculous example, suppose that $\text{Enc}_k(m) = m$ for all k – that is, the "encryption" scheme ignores the key completely and just sends the message. This is clearly not perfectly secret.

Less extreme examples can be constructed as well. By Shannon's Theorem, if $|\mathcal{K}| = |\mathcal{M} = |\mathcal{C}|$ and keys are sampled uniformly at random, then Π is perfectly secret if and only if for each m and c, there is a unique k with $\text{Enc}_k(m) = c$. So any scheme where two different keys encrypt the same message to the same ciphertext would be a counterexample.

Problem 2.8: We went over this in class, but here is the argument again. As the hint says, if Π is not perfectly secret then there are messages m_0, m_1 and a ciphertext c with $\Pr[C = c|M = m_0] > \Pr[C = c|M = m_1]$. Now consider the adversary $\mathcal{A}$ which chooses m_0 and m_1 as input. If $\mathcal{A}$ is given back the ciphertext c, then he guesses that $b = 0$, otherwise he makes a random guess. It is intuitively clear that $\mathcal{A}$ will succeed with probability greater then 1/2. To do the formal calculation, let $\Pr[C = c|M = m_0] = p$ and $\Pr[C = c|M = m_1] = q$. Then the success probability of $\mathcal{A}$ is

$$1 \times p + 0 \times q + 1/2 \times (1 - p - q) = 1/2 + 1/2(p - q).$$

The first two terms reflect that $\mathcal{A}$ either always succeeds or always fails when given c, depending on whether $M = m_0$ or $M = m_1$ and the third term reflects the random guess when $C \neq c$. Since $p > q$, the success probability is strictly larger than 1/2.

Problem 2.9: The key to this problem is the fact that decryption is deterministic: Given the key k and the ciphertext c, $\mathrm{Dec}_k(c)$ must equal the original message m. So if m and m' are *different* messages encrypted with the same key k, then their corresponding ciphertexts *must* be different. This means that, if $c = c'$, we have

$$\Pr[M = m \wedge M' = m' | C = c \wedge C' = c'] = 0 \neq \Pr[M = m \wedge M' = m'].$$

Note that we cannot make a similar argument by taking $m = m'$ and $c \neq c'$. Since encryption is allowed to be randomized, we cannot conclude that different ciphertexts must be the encryption of different plaintexts.

Modern Cryptography
600.442
Homework 2 Solutions

Dr. Christopher Pappacena

Due September 19, 2013

Problem 3.1: To prove part (a), fix a polynomial $p(n)$ and let $q(n) = 2p(n)$. Then for n large enough, we have

$$\mathtt{negl}_1(n) < \frac{1}{q(n)} \text{ and } \mathtt{negl}_2(x) < \frac{1}{q(n)}.$$

This says that

$$\mathtt{negl}_3(n) = \mathtt{negl}_1(n) + \mathtt{negl}_2(n) < \frac{1}{q(n)} + \frac{1}{q(n)} = \frac{2}{q(n)} = \frac{1}{p(n)}$$

so $\mathtt{negl}_3(n)$ is negligible.

For part (b), fix a polynomial $f(n)$ and let $q(n) = p(n)f(n)$. Then for n sufficiently large we have

$$\mathtt{negl}_1(n) < \frac{1}{q(n)}.$$

This says that

$$\mathtt{negl}_4(n) = p(n)\,\mathtt{negl}_1(n) < \frac{p(n)}{q(n)} = \frac{1}{f(n)}.$$

So $\mathtt{negl}_4(n)$ is negligible.

Problem 3.3: The point of this exercise is that the adversary can reliably choose messages m_0 and m_1 whose ciphertexts will have different lengths, so that he can determine which message was encrypted just by looking at the length.

The hint gives us the details to back up this intuition. As it says, let $q(n)$ be a polynomial upper bound on the length of a ciphertext when a single bit is encrypted with Π. Choose m_0 to be either 0 or 1, and let m_1 be a message of length $q(n) + 2$. Since there are $2^{q(n)+2}$ messages with this length, and at most $2^{q(n)}$ ciphertexts of length $q(n)$, the probability that the ciphertext corresponding to m_1 has length strictly greater than $q(n)$ is at least $3/4$. So, choosing 0 when the ciphertext has length $\leq q(n)$ and 1 otherwise will succeed with probability at least 3/4, so this adversary always has a non-negligible advantage.

Problem 3.4: The idea of this problem is to use a technique called *padding* so that all messages have the same length. We could define a new encryption scheme $\Pi' = (\mathrm{Gen}', \mathrm{Enc}', \mathrm{Dec}')$ by setting $\mathrm{Gen}' = \mathrm{Gen}$, $\mathrm{Dec}' = \mathrm{Dec}$, and defining Enc' by

$$\mathrm{Enc}'_k(m) = \mathrm{Enc}_k(m||0^{\ell(n)-|m|}).$$

That is, we "pad" m to length exactly $\ell(n)$ by filling it with zeroes as necessary.

As stated, this almost works, but not quite. The problem is that multiple messages encrypt to the same ciphertext. Given a ciphertext c, if we compute $\mathrm{Dec}'_k(c) = m||0^{\ell(n)-|m|}$ then, without knowing the length of m, we don't know how many zeroes to ignore. This violates the basic requirement that decryption be deterministic and always succeed. Of course, the encryption scheme can send the length of m as part of the ciphertext, but this defeats the entire purpose of padding out the message!

To fix this we need to send the *encryption* of the length of the message. A simple but inefficient way to do this is to encode $|m|$ as an integer of binary length exactly $\ell(n)$ and define Enc' by

$$\mathrm{Enc}'_k(m) = (\mathrm{Enc}_k(|m|), \mathrm{Enc}_k(m||0^{\ell(n)-|m|})).$$

Now, given a ciphertext c, we decrypt each of the two components to recover both the message and its length. The only flaw with this system that ciphertexts are twice a large as plaintexts.

We can improve this if we work a little harder. First note that the number of bits needed to encode the length of a message m is at most $\lceil \log_2(\ell(n)) \rceil$. We modify the message space of Π' to only encrypt messages of length at most $\ell(n) - \lceil \log_2(\ell(n)) \rceil$. Given a message m, we pad it out with as many zeroes as necessary to give it length $\ell(n) - \lceil \log_2(\ell(n)) \rceil$, then append the length of m in binary, using exactly $\lceil \log_2(\ell(n)) \rceil$ bits. Upon decrypting the ciphertext, we read off the length of the message and we know how many zeroes at the end of the message to ignore, if any.

For example, suppose that $\ell(n) = 31$. We need 5 bits to encode an integer between 0 and 31 so we reduce the size of our message space down to $31 - 5 = 26$ bits. Suppose our actual message m has length 21. We write $21 = 10101$ in binary and set

$$c = \mathrm{Enc}'_k(m) = \mathrm{Enc}(m||00000||10101).$$

Decrypting c, we read the last 5 bits to see that the message m has length 21, and we only look at the first 21 bits to recover m.

This system, while more complicated, is also more efficient since, as a function of n, the ratio of the size of the messages to the size of the ciphertexts approaches 1 as $n \to \infty$.

TURN IN: 2.2, 2.3, 2.5, 2.9, 3.4, 3.6

Modern Cryptography
600.442 Fall 2013
Homework 3 Solutions

Problem 3.6: (a) To clarify notation in this problem, $0^{|s|}$ means a run of zeroes of length $|s|$. In other words, if s is an n-bit input, then $s||0^{|s|}$ means the $2n$-bit input obtained by tacking on n zeroes to the end of s. If we define $G'(s) = G(s||0^{|s|})$ then G' is not necessarily pseudorandom. Intuitively, a distinguisher which can tell the output of G' apart from random is able to distinguish the output of G from random, provided the second half of the seed is all zeroes. But this represents a negligible proportion of the possible input seeds to G and does not produce a contradiction.

More formally, let H be a pseudorandom function and define G by $G(x) = H(x)$, unless $|x| = n = 2m$ is even and $x = s||0^{|s|}$ for some $s \in \{0,1\}^m$. In this case, we set $G(x) = x$. The proportion of inputs for which $H(x) \neq G(x)$ is $2^{-n/2}$, which is a negligible function of n. It follows from this that G is pseudorandom. On the other hand, $G'(s) = s||0$ for all s and this function is definitely *not* pseudorandom.

(b) In this case, G' must be a pseudorandom generator. We will demonstrate this by reducing a distinguisher for G' to a distinguisher for G. As above let $n = 2m$. By the definition of G', the length of the output of G' is $\ell(m)$. By assumption $\ell(m) > 2m = n$ so that G' does have the expansion property.

Let D be a distinguisher for G' with advantage $\epsilon(n)$, so that

$$|\Pr[D(G'(s)) = 1] - \Pr[D(r) = 1]| = \epsilon(n),$$

where the probabilities are taken over all $s \in \{0,1\}^n$ and r in $\{0,1\}^{\ell(m)}$. The key observation is that, as s ranges uniformly over $\{0,1\}^n$, then $s_1 \ldots s_m$ ranges uniformly over $\{0,1\}^m$. This means that

$$|\Pr[D(G(s)) = 1] - \Pr[D(r) = 1]| = \epsilon(n)$$

as s ranges over $\{0,1\}^m$ and r ranges over $\{0,1\}^{\ell(m)}$.

In other words, the distinguisher D, run on G with input length m, can tell the output of G from random with advantage $\epsilon(2m)$. Since G is pseudorandom, $\epsilon(2m)$ is negligible. It is easy to see that this implies that $\epsilon(m)$ is negligble as well. Hence G' is pseudorandom.

Problem 3.10: This problem is very similar to the example we gave in class. The observation which works here is that, if x and y are any two inputs, then

$$F_k(x) \oplus F_k(y) = G'(k) \oplus x \oplus G'(k) \oplus y = x \oplus y.$$

In other words, for any key k, the function F_k has the property that the XOR of any two outputs is equal to the XOR of the corresponding inputs. This is a highly nonrandom property and so a distinguisher can confirm that it is sampling from F_k after only a few evaluations.

Problem 3.13: Define the function F' by $F'_k(x) = F_k(x) \oplus F_k(k)$. Notice that this function has the property that $F'_k(k) = 0^{|k|}$. To see that it is a permutation, suppose that $F'_k(x) = F'_k(y)$. Then $F_k(x) \oplus F_k(k) = F_k(y) \oplus F_k(k)$, so $F_k(x) = F_k(y)$. Since F_k is a permutation, $x = y$ and F'_k is a permutation. It is pseudorandom because F'_k and F_k differ by a fixed element of $\{0,1\}^n$, so any distinguisher which can tell F'_k from random can also tell F_k from random.

To see that F'_k is not a *strong* pseudorandom permutation, note that a distinguisher can query the decryption oracle to get $F_k^{-1}(0^{|k|}) = k$. Since F is a known function, once the distinguisher has k he can easily test if the function he has oracle access to is F_k or a random permutation.

Problem 3.15: (a) This scheme does not necessarily have indistinguishable encryptions in the presence of an eavesdropper becuase the values $G(k)$ and $G(k+1)$ may be correlated to each other. If this happens then an attacker may be able to distinguish the encryption of $m_0||m_0$ from $m_0||m_1$. For a concrete example of this, let G' be a pseudorandom generator and let $G(s) = G'(s_1 \dots s_{n/2})$. (This is the generator of Problem 3.6(b)). Then for many values of k, $G(k) = G(k+1)$ and an attacker can easily tell encryptions of $m_0||m_0$ and $m_0||m_1$ apart.

Note that even if G is chosen so that Π *does* have indistinguishable encryptions in the presence of an eavesdropper, it is not CPA-secure because encryption is deterministic.

(b) This scheme is completely insecure: Since the ciphertext contains the random seed r, anyone can decrypt!

(c) This scheme has indistinguishable encryptions in the presence of an eavesdropper – an adversary $\mathcal{A}$ who can distinguish encryptions of m_0 and m_1 with non-negligible probability will learn the value of $F_k(0)$ with non-negligible probability. But then $\mathcal{A}$ can distinguish F_k from a random function with non-negligible probability. However it is *not* CPA secure since encryption is completely deterministic.

(d) This scheme is CPA secure. If we replace F_k by a random function then the only way an attacker can succeed is if one of the random seeds he sees when requesting encryptions of chosen plaintexts is $r-1$, r, or $r+1$. This happens with negligible probability. Now, if we replace a truly random function with a pseudorandom function, then a distinguisher can use an adversary who can break Π to distinguish F_k from a random function. Since F_k is pseudorandom this says that Π is secure.

Problem 3.21: Here is one way to solve the problem using the suggested hint: Define $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ as follows. $\text{Gen}(n)$ returns two keys $k_1, k_2 \leftarrow \{0,1\}^n$. Given a message m, $\text{Enc}_k(m)$ selects $r \leftarrow \{0,1\}^n$ and sets

$$\text{Enc}_k(m) = (\text{Enc}_1(k_1, r), \text{Enc}_2(k_2, m \oplus r)).$$

Decryption is accomplished by $\text{Dec}_k(c_1, c_2) = \text{Dec}_1(k_1, c_1) \oplus \text{Dec}_2(k_2, c_2)$.

Intuitively, if an adversary $\mathcal{A}$ is able to break Π_1 then he can potentially learn $m \oplus r$, but since r is random he learns nothing about m. Similarly if $\mathcal{A}$ is able to break Π_2, he can potentially learn r but this too reveals nothing about m. As long as $\mathcal{A}$ is not able to break both Π_1 and Π_2, the system is secure.

To make this a formal proof, let $\mathcal{A}$ be an adversary who can break Π with advantage $\epsilon(n)$ and let $\mathcal{A}'$ be an adversary trying to break one Π_2. Note that we give $\mathcal{A}'$ oracle access to each of Π_1 and Π_2. $\mathcal{A}'$ calls $\mathcal{A}$ as a subroutine and receives messages m_0 and m_1. Next, $\mathcal{A}'$ chooses a random r and presents $r \oplus m_0$ and $r \oplus m_1$ to Π_2, getting back $\text{Enc}_2(k_2, r \oplus m_b)$. $\mathcal{A}'$ also uses oracle access to Π_1 to get $\text{Enc}_1(k_1, r)$, and gives the ciphertext $(\text{Enc}_1(k_1, r), \text{Enc}_2(k_2, r \oplus m_b))$ to $\mathcal{A}$. Finally, $\mathcal{A}'$ returns the value returned by shA.

The probability that $\mathcal{A}'$ succeeds in the indistinguishability experiment is exactly $\epsilon(n)$, so if Π_2 is secure, then so is Π. On the other hand, if $\mathcal{A}'$ is trying to break Π, there is an analogous algorithm he can run with $\mathcal{A}$ as a subroutine, where the ciphertext he presents to $\mathcal{A}$ is $(\text{Enc}_1(k_1, r \oplus m_b), \text{Enc}_2(k_2, r))$. Note that if we set $s = r \oplus m_b$, then this ciphertext is exactly $(\text{Enc}_1(k_1, s), \text{Enc}_2(k_2, s \oplus m_b)$, which is a valid instance of Π. So the probability that $\mathcal{A}'$ breaks Π_1 in this case is also $\epsilon(n)$, showing that if Π_1 is CPA-secure, then so is Π.

Modern Cryptography
600.442 Fall 2013
Homework 4 Solutions

Problem 5.4: Note that r rounds of key mixing in a row is really just a single, more complicated key mixing. Similarly, r rounds of substitution in a row are a single substitution and r consecutive permutations are a single permutation. This substitution-permutation network is only one round long! Using the attack in the book, an adversary can easily recover the round key used to encrypt which is enough to read the message.

Problem 5.5: In a Feistel network, the relationship between the pair (L_{i+1}, R_{i+1}) and (L_i, R_i) is

$$L_{i+1} = R_i, \qquad R_{i+1} = L_i \oplus f(R_i).$$

(a) If $f(R_i) = 0$ for all possible inputs, then $R_{i+1} = L_i$ and we see that one round of the Feistel network simply interchanges the left and right halves. So, if r is even then $(L_r, R_r) = (L_0, R_0)$ and if r is odd then $(L_r, R_r) = (R_0, L_0)$. In other words, the result is to do nothing if r is even, and to switch the two halves when r is odd.

(b) If $f(R_i) = R_i$ for all i, then we can simply track what happens through the first few rounds:

- $(L_1, R_1) = (R_0, L_0 \oplus R_0)$.
- $(L_2, R_2) = (R_1, L_1 \oplus R_1) = (L_0 \oplus R_0, R_0 \oplus L_0 \oplus R_0) = (L_0, R_0)$.

The pattern then repeats. So in this case we have $(L_r, R_r) = (L_0, R_0)$ when r is even and $(L_r, R_r) = (R_0, L_0 \oplus R_0)$ when r is odd.

Problem 5.6: The key observation is that, if $f_i(k, x)$ is the value of the DES mangler function at round i with input x and key k, then $f_i(k, x) = f_i(\bar{k}, \bar{x})$. This is because the round key just selects bits from the master key and $E(\bar{x}) = \overline{E(x)}$. So, the two values $E(x) \oplus k_i$ and $E(\bar{x}) \oplus \bar{k}_i$ agree, and the rest of the round function depends only on this quantity.

If the state of the DES Feistel network at round i is (L_i, R_i), then the state at round $i+1$ is $(L_{i+1}, R_{i+1}) = (R_i, L_i \oplus f_i(k, R_i))$. This means that if the state of DES is $(\overline{L_i}, \overline{R_i})$ in round i, its state at round $i+1$ will be

$$(\overline{R_i}, \overline{L_i} \oplus f_{\bar{k}}(\bar{k}, \overline{R_i})) = (\overline{R_i}, \overline{L_i} \oplus f_k(k, R_i)) = (\overline{L_{i+1}}, \overline{R_{i+1}}).$$

So complementing the plaintext and the key complements every round of DES, including the final output: $\mathrm{DES}_{\bar{k}}(\bar{x}) = \overline{\mathrm{DES}_k(x)}$.

Problem 5.7: The plaintexts don't have to be chosen *that* carefully – they just have to be complements of each other, for example $m_0 = 0^{56}$ and $m_1 = 1^{56}$. Let c_0 and c_1 be the corresponding ciphertexts and let $\mathcal{K}_0$ be the set of DES keys whose first bit is 0. Note that $|\mathcal{K}_0| = 2^{55}$ and $\mathcal{K}_0 \cup \overline{\mathcal{K}}_0 = \mathcal{K}$. (That is, every DES key or its complement is contained in $\mathcal{K}_0$.

Now we encrypt m_0 with every possible key in $\mathcal{K}_0$. If the actual key k is in $\mathcal{K}_0$, then we will see $c_0 = \mathrm{DES}_k(m_0)$ and know the correct key is k. If the actual key k is in $\overline{\mathcal{K}}_0$, then at some point we will encrypt m_0 with the key $\bar{k}$. When this happens, we have

$$\mathrm{DES}_{\bar{k}}(m_0) = \mathrm{DES}_{\bar{k}}(\bar{m}_1) = \overline{DES_k(m_1)} = \overline{c_1}.$$

Since we know c_1 we will see this happen and know the correct key is $\bar{k}$.

Problem 5.8: As described on page 172 of the text, we can invert the action of a Feistel network by observing that

$$L_{i-1} = R_i \oplus f(L_i), \qquad R_{i-1} = L_i.$$

This means that the inverse of an r-round Feistel network looks like the original Feistel network, with two exceptions:

- The round functions are applied in the opposite order, from f_r down to f_1.
- The roles of the left and right halves are interchanged.

In DES, the second issue is resolved by not performing the last swap. This keeps the left and right halves "on cut" and means that DES^{-1} is indeed a Feistel network, with the round functions given in opposite order. Since the only thing that changes from one round to the next is the addition of subkey, this means that we can implement DES^{-1} by just using the regular DES subkeys in reverse order. In fact, more is true. Since the key schedule for DES operates by left cyclic rotation of the key bits, we can produce the key schedule for decryption by changing to a right cyclic rotation.

Problem 5.9: (a) Since DES decryption is DES encryption with the order of the subkeys reversed, and since every subkey for the key 0^{56} is 0^{48}, we see that encryption and decryption are the same. This says that $\mathrm{DES}_k(\mathrm{DES}_k(x)) = x$ when $k = 0^{56}$. This property is a security threat in the presence of a chosen-plaintext adversary – the adversary can send in ciphertexts to the encryption oracle and receive back the corresponding plaintexts!

(b) Just as the all-zeroes key produces the same round key in either order, the all-ones key 1^{56} has the property that every round key is 1^{48}. So this is another weak key for DES. Note we can also get this using the complementary property of DES. Denote the all zeroes and all ones keys by 0 and 1 for simplicity. Then

$$\mathrm{DES}_1(\mathrm{DES}_1(x)) = \overline{\mathrm{DES}_0(\overline{\mathrm{DES}_1(x)})} = \overline{\mathrm{DES}_0(\mathrm{DES}_0(\bar{x}))} = \bar{\bar{x}} = x.$$

Now we use the fact that the two halves of the key schedule are independent. The DES essentially derives two 24-bit round keys from the two 28-bit halves of the key independently. From this we see that the keys $0^{28}||1^{28}$ and $1^{28}||0^{28}$ are also weak keys for DES.

(c) These keys do not present any security problem – the probability that any one of them is selected at random from $\{0,1\}^{56}$ is 2^{-54}. If we wanted to, we could add a subroutine to our key generation algorithm to check if one of these four weak keys were selected and generate a new one if necessary. This reduces the size of the DES key space to $2^{56} - 4$ which does not appreciably reduce the security of DES.

Modern Cryptography
600.442 Fall 2013
Homework 5 Solutions

Problem 4.3: An adversary $\mathcal{A}$ can use his oracle access obtain the tags for two messages, say (m_1, m_2) and (m'_1, m'_2). Suppose the first tag is $t = (F_k(m_1), F_k(F_k(m_2)))$ and the second tag is $t' = (F_k(m'_1), F_k(F_k(m'_2)))$. From these, $\mathcal{A}$ can produce the valid tag $(F_k(m_1), F_k(F_k(m'_2)))$ for the message (m_1, m'_2) and succeed with probability 1.

Problem 4.4: (a) Suppose we ask for the tag t of the message $m = (m_1, \ldots, m_\ell)$. Then we can insert an arbitrary message block m' into two locations of the message without affecting the tag, since the new tag will be $t \oplus F_k(m') \oplus F_k(m') = t$. So, for example, t is a valid tag for $(m', m_1, \ldots, m_\ell, m')$.

(b) Since the tag does not track the location of the blocks in the message we can change the ordering of the blocks without affecting the tag. For example, if we receive (r, t) as the tag for (m_1, m_2), then it will also be the tag for the message (m_2, m_1).

(c) Since r can be arbitrary, if m is a message of length $n/2$ then $(\langle 1 \rangle \| m, 0)$ is a valid tag for m.

Problem 4.5: (a) To provide an analogue of Definition 4.2, we first need to provide an analogue of the message authentication experiment. In the new experiment, the adversary $\mathcal{A}$ is given oracle access to $\text{Mac}_k(\cdot)$ and $\text{Verify}_k(\cdot)$. $\mathcal{A}$ is required to output a (m, t) for a message m which he has not previously authenticated using access to the Mac oracle. Note that it should be permissible for $\mathcal{A}$ to submit as his response a pair (m, t) which he has submitted to the Verify oracle, because this still implies that $\mathcal{A}$ has successfully forged the tag for m.

The formal definition is that Π is secure if, for all PPT adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ successfully produces a forgery (m, t) is a negligible function of the security parameter n.

The real-world interpretation of access to a Verify oracle is an adversary who is able to send messages to the recipient and observe if the recipient accepts them as valid. There are applications where a recipient will respond to an invalid message by requesting a retransmission, and in these settings an adversary is essentially given access to the Verify oracle.

(b) Suppose that Π is a secure MAC using Definition 4.2. Since tags are unique, there are only two ways that an adversary $\mathcal{A}$ can present a valid pair (m,t) to the Verify oracle: $\mathcal{A}$ could have queried the MAC oracle for $t = \text{Mac}_k(m)$, in which case $\mathcal{A}$ already knows that (m,t) is valid, or $\mathcal{A}$ is producing a valid pair (m,t) which requires breaking the MAC. Since the MAC is existentially unforgeable, we conclude that the event that $\mathcal{A}$ gets an answer of 1 from the Verify oracle is negligible. In effect, the Verify oracle is useless to $\mathcal{A}$ - it will only ever return 0. So the MAC remains secure even when access to the Verify oracle is given.

(c) I confess I struggled with this problem for a while and I think that it is because it is a little-bit ill-formed. As far as I can see, a solution runs along the following lines. Suppose that we take a MAC with unique tags and add some extra bits to the tag which are not deterministic, to make a MAC without unique tags. Let us write the tag as (t,r) to represent this, where the t comes from the deterministic MAC and the r represents the non-deterministic part. Suppose also thar r is small, say r is $\log n$ bits, so exhausting r is polynomial time. If $\mathcal{A}$ gets a valid pair (m,t,r) from the MAC oracle, then $\mathcal{A}$ can use the verify oracle, changing the random bits, until he finds r' with (m,t,r') valid. He then returns (m,t,r') for his answer. The reason I don't like this solution is that the adversary is forging a message he already knows how to authenticate via the original tag (t,r). But I don't know any other way to do this problem offhand. I will keep searching for a better solution.

Problem 4.9: (a) Suppose that the adversary $\mathcal{A}$ asks for a message tag for message $(m_1,\ldots,m_\ell)$ and receives (t_0,t_ℓ). By the definition of CBC-MAC, we have that $t_{i+1} = F_k(t_i \oplus m_{i+1})$ for all $0 \le i \le \ell - 1$. If we let $m_1' = t_0 \oplus m_1$ and $t_0' = 0$, then $t_1 = F_k(t_0 \oplus m_1) = F_k(t_0' \oplus m_1')$. So $(0,t_\ell)$ is a valid tag for $(m_1', m_2, \ldots, m_\ell)$.

(b) To forge a message, $\mathcal{A}$ asks for tags for two messages $(m_1,\ldots,m_\ell)$ and $(m_1',\ldots,m_\ell')$. Call these tags $(t_1,\ldots,t_\ell)$ and $(t_1',\ldots,t_\ell')$. Now $\mathcal{A}$ can choose any index i and define

$$m_i'' = t_{i-1} \oplus t_{i-1}' \oplus m_i'.$$

By definition we have

$$F_k(t_{i-1} \oplus m_i'') = F_k(t_{i-1}' \oplus m_i') = t_i'.$$

So, $(t_1,\ldots,t_{i-1},t_i',\ldots,t_\ell')$ is a valid tag for $(m_1,\ldots,m_{i-1},m_i'',m_{i+1}',\ldots,m_\ell')$.

Problem 4.11: (a) Suppose that $\mathcal{A}$ is able to find a pair x, x' with $H^{s_1,s_2}(x) = H^{s_1,s_2}(x')$. Then by definition of H, this means that $H_1^{s_1}(x) = H_1^{s_1}(x')$ and $H_2^{s_2}(x) = H_2^{s_2}(x')$. So $\mathcal{A}$ has found a collision in both H_1 and H_2. As long as one of H_1 and H_2 is secure this is a contradiction.

Note that the above argument can be turned into a formal proof of security: An adversary $\mathcal{A}'$ who is trying to find a collision in H_1, say, can use $\mathcal{A}$ as a subroutine to break H_1 with the same advantage that $\mathcal{A}$ has in finding a collision in H. So if H_1 is collision resistant, H must be also.

(b) If H does not have second pre-image resistance then given $s = (s_1, s_2)$ and x, we can find x' with $H^{s_1,s_2}(x') = H^{s_1,s_2}(x)$. But this says that $H_i^{s_i}(x') = H_i^{s_i}(x)$ for $i = 1, 2$. So, if one of H_1 or H_2 is second pre-image resistant, then so is H.

However, it may be true that H is not pre-image resistant even if one of H_1 or H_2 is pre-image resistant. For example, suppose that H_1 is a pre-image resistant hash function with the property that $H_1^{s_1}(x) = H_1^{s_1}(x_{\leq n})$, where the notation $x_{\leq n}$ means the first n bits of x. Now let $H_2^{s_2}(x) = x_{\leq n}$ for all seeds s_2. Clearly H_2 is not pre-image resistant. For the combined hash function H, if we are given $y = H_1^{s_1}(x)||H_2^{s_2}(x) = H_1^{s_1}(x)||x_{\leq n}$, we can read off $x' = x_{\leq n}$ from y and then $H^s(x') = y$.

Problem 4.19: Recall that in encrypt-and-authenticate, the transmitted value is (c, t) where $c = \text{Enc}_{k_1}(m)$ and $t = \text{Mac}_{k_2}(m)$. Suppose that the adversary $\mathcal{A}$ presents plaintexts m_0 and m_1 and receives back (c, t) where $c = \text{Enc}_{k_1}(m_b)$ and $t = \text{Mac}_{k_2}(m_b)$. Even though the encryption may be randomized, since the tag is unique $\mathcal{A}$ can ask for authenticated encryptions of m_0 and m_1 again, and check which one has a tag that matches t. This will reveal the value of b even though the ciphertext portion of the response will be different.

Modern Cryptography
600.442
Homework 6 Solutions

November 2, 2013

Problem 6.4: Let $\mathcal{A}$ be an adversary which inverts g with probability $\epsilon(n)$. Given $f(x)$, $\mathcal{A}$ selects r uniformly at random and inverts $g(x,r)$. Using ideas similar to the proof of the existence of hardcore predicates, there is a subset S_n of $\{0,1\}^n$ of size at least $\epsilon(2n)2^n/2$ such that $\mathcal{A}$ successfully finds a preimage (x', r) with probability $\geq \epsilon(2n)/2$. Note that x' is a preimage of $f(x)$ so we have inverted f with probability at least $\epsilon(2n)/2$. This says that $\epsilon(2n)$ is a negligible function of n, so that $\epsilon(n)$ is as well, and g is one-way.

Problem 6.5: In general, G is *not* a pseudorandom generator, because f can induce a very non-random distribution on outputs. For example, suppose that f is a one-way function with the property that the first bit of $f(x)$ is 0 for all x. (Given any one-way function g, we can make a one-way function f with this property. We simply define $f(x)$ by evaluating $g(x)$ and overwriting the first bit to 0.) We can define a distinguisher D for the output of G by returning 1 if the first bit is 0, otherwise returning 0. The probability that $D(G(s)) = 1$ is 1, while the probability that $D(r) = 1$ is 1/2. Hence

$$|Pr[D(G(s)) = 1] - Pr[D(r) = 1]| = \frac{1}{2}$$

is non-negligible, and G is not pseudorandom.

Problem 6.7: The function $f(f(x))$ is *not* a one-way function in general. To see why, suppose that $g(x)$ is a one-way function and define a function f by $f(x,y) = (g(y), 0)$. This is still a one-way function, since finding a preimage for f will produce a preimage for g. However, if we compose f with itself we get

$$f(f(x,y)) = f((g(y), 0)) = (g(0), 0)$$

which is constant. So any (x', y') is a preimage and $f(f(x,y))$ is easy to invert.

The function $g(x) = (f(x), f(f(x))$ is a one-way function, because any preimage for g gives a preimage for f. So, if $\mathcal{A}$ is an algorithm that can find a preimage for g, we can construct an algorithm $\mathcal{A}'$ which computes a preimage for f as follows: Given $y = f(x)$, $\mathcal{A}'$ computes $y' = f(y)$ and calls $\mathcal{A}$ as a subroutine with input (y, y'). Then $\mathcal{A}'$ returns the answer x' provided by $\mathcal{A}$. If $g(x') = (y, y')$, then $f(x') = y$ so $\mathcal{A}'$ succeeds with the same probability as $\mathcal{A}$. This shows that g is one-way if f is.

Problem 6.10: Let $\mathcal{A}$ be an algorithm which can invert f with probability $\epsilon(n)$. Given $f(x)$, we run $\mathcal{A}$ to get a value x' and check if $f(x) = f(x')$. If so, then $x = x'$ since f is one-to-one and we can compute $\mathrm{hc}(x)$. If $x \neq x'$ then we guess $\mathrm{hc}(x)$ uniformly. This strategy will find $\mathrm{hc}(x)$ with probability $1/2 + \epsilon(n)/2$. Since hc is a hardcore predicate for f, $\epsilon(n)$ is negligible. This in turn says that f is one-way.

Problem 6.13 For $0 \leq i \leq n$, we define a probability distribution H_n^i on $\{0,1\}^{n\ell(n)}$. We choose i values from $\{0,1\}^{\ell(n)}$, say $r_1, \dots, r_i$, and we choose $n - i$ values from $\{0,1\}^n$, say $s_{i+1}, \dots, s_n$, and return the value

$$(r_1, \dots, r_i, G(s_{i+1}), \dots, G(s_n)).$$

Note that H_n^0 is the distribution induced by $\tilde{G}$ and H_n^n is the uniform distribution.

Let D be a distinguisher for$\tilde{G}$ which succeeds with advantage $\epsilon(n)$, We define a distinguisher D' for G as follows. Given a string $y \in \{0,1\}^{\ell(n)}$, D' selects $i \leftarrow \{0, dots, n-1\}$, chooses i uniformly random values from $\{0,1\}^{\ell(n)}$, chooses $n - i - 1$ random values from $\{0,1\}^n$, and presents D with the value

$$(r_1, \dots, r_i, y, G(s_{i+2}), \dots, G(s_n)).$$

Note that if $y = G(x)$ for some x, then D is given a sample from H_n^i and if y is random then D is given a sample from H_n^{i+1}. D' returns the same answer as D.

As in class the probability that D' returns 1 when $y = G(x)$ is

$$Pr[D'(G(x)) = 1] = \frac{1}{n} \sum_{i=1}^{n} \Pr_{H_n^i}[D(z) = 1]$$

and the probability that D' returns 1 when y is random is

$$Pr[D'(y) = 1] = \frac{1}{n} \sum_{i=0}^{n-1} \Pr_{H_n^i}[D(z) = 1].$$

Hence we have

$$|\Pr[D'(G(x)) = 1] - \Pr[D'(y) = 1]| = \frac{1}{n} |\Pr[D(\tilde{G}(s)) = 1] - \Pr[D(r) = 1]|.$$

So D' succeeds with probability $\epsilon(n)/n$ and $\epsilon(n)$ is negligible.

Problem 6.20: Let $\mathcal{A}$ be an algorithm that can invert G with probability $\epsilon(n)$. We use $\mathcal{A}$ to build a distinguisher D for G as follows. Given a string $y \in \{0,1\}^{n+1}$, D calls $\mathcal{A}$ as a subroutine on input y and receives a value $x \in \{0,1\}^n$. If $G(x) = y$ then D returns 1; otherwise D selects and returns $b \leftarrow \{0,1\}$ uniformly at random.

We have

$$\Pr[D(G(x)) = 1] = \frac{1}{2} + \frac{\epsilon(n)}{2},$$

since D returns 1 with probability 1 when $\mathcal{A}$ inverts G and with probability $1/2$ when $\mathcal{A}$ fails to invert f. On the other hand, we have

$$\Pr[D(y) = 1] = \frac{1}{2} + \frac{\epsilon(n)}{4}.$$

This is because a random $y \in \{0,1\}^{n+1}$ is in the image of G with probability $1/2$. If y is not in the image of G, then D returns 1 with probability $1/2$ and if y is in the image of G, D returns 1 with probabillity $1/2 + \epsilon(n)/2$.

Combining these we have

$$|\Pr[D(G(x)) = 1] - \Pr[D(y) = 1]| = \frac{\epsilon(n)}{4}.$$

Since G is a pseuodrandom generator, $\epsilon(n)$ is negligible and so G is a one-way function.

Modern Cryptography
600.442
Homework 4 Solutions

November 10, 2013

Problem 7.15: Let $\mathcal{A}$ be an algorithm that can solve the discrete logarithm problem with advantage $\epsilon(n)$. We construct an algorithm $\mathcal{A}'$ which uses $\mathcal{A}$ as a subroutine to solve the CDH problem.

Given (G, q, g, h_1, h_2), $\mathcal{A}'$ calls $\mathcal{A}$ as a subroutine with input (G, q, g, h_1) and receives a value x_1. $\mathcal{A}'$ checks if $h_1 = g^{x_1}$ and if so returns $h = h_2^{x_1}$. If $h_1 \neq g^{x_1}$ then $\mathcal{A}'$ returns a random element of G.

The probability that $\mathcal{A}$ succeeds for an instantance of the CDH problem is $\epsilon(n)$. Conditioned on this event, $\mathcal{A}'$ succeeds with probability 1. So, the total probability that $\mathcal{A}'$ succeeds is at least $\epsilon(n)$, since $\mathcal{A}'$ can also get the answer right when $\mathcal{A}$ fails by random guessing. Since the CDH problem is hard, $\epsilon(n)$ is negligible and so the discrete log problem is hard too.

Problem 7.16: Assume that the DDH problem is hard. Let $\mathcal{A}$ be an algorithm that can solve the CDH problem with advantage $\epsilon(n)$. Let $\mathcal{A}'$ be an algorithm which, on input (G, q, g, g^x, g^y, h), calls $\mathcal{A}$ as a subroutine with input (G, q, g, g^x, g^y). If $\mathcal{A}$ returns h, then $\mathcal{A}'$ returns 1, otherwise $\mathcal{A}'$ returns 0.

We have

$$|\Pr[\mathcal{A}'(g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}'(g^x, g^y, h) = 1]| = \epsilon(n).$$

So $\epsilon(n)$ is negligible and the harness of the DDH problem implies the hardness of the CDH problem.

Problem 7.18: This problem is not hard. Given $x \in \mathbb{Z}_{p-1}^*$, it is possible to compute z such that $xz = 1 \pmod{p-1}$ in polynomial time using the extended GCD algorithm. With z in hand, it is easy to compute

$$y^z = g^{xz} = g^1 = g \pmod{p}.$$

So finding g from y can be solved with probability 1.

Problem 7.20: We define a family of one-way permutations as follows.

- Gen(n) calls $\mathcal{G}_1(n)$ to obtain $(\mathbb{Z}_p^*, p-1, g)$, where g is a fixed generator of $\mathbb{Z}_p^*$. We set $I = (\mathbb{Z}_p^*, p-1, g)$ and $D_I = \mathbb{Z}_p^*$.
- Samp(I) chooses a random element x of $\mathbb{Z}_p^*$ by selecting $x \leftarrow \{1, \ldots, p-1\}$.
- On input I, $f_I(x) = g^x \in \mathbb{Z}_p^*$.

To see that f_I is a permutation, note that if $g^x = g^y \pmod{p}$, then $x = y \pmod{p-1}$ by Lagrange's Theorem. Since x and y both come from $\{1, \ldots, p-1\}$, we must have $x = y$.

The inversion problem for f is precisely the discrete logarithm problem for $\mathbb{Z}_p^*$. So, if the discrete logarithm problem is hard, f is a one-way permutation.

Problem 9.1: (a) The indistinguishability experiment for Π' would involve giving $\mathcal{A}$ the transcript of the key exchange, $\mathcal{A}$ picking ciphertexts m_0 and m_1 and receiving the ciphertext corresponding to $\text{Enc}_k(m_b)$ for random b. Then $\mathcal{A}$ must guess b, and the system is secure if the probability that $\mathcal{A}$ guesses b correctly is $1/2 + \texttt{negl}(n)$ for some negligible function of n.

(b) Define a third encryption scheme $\tilde{\Pi}$ by having the recipients execute the key exchange, but then use a completely independent key for the encryption. (Never mind that in the real world they wouldn' have a key to use.) The point is that an adversary $\mathcal{A}$ can only break $\tilde{\Pi}$ if he can break the private-key encryption scheme, since the key exchange contributes no information. Since the encryption scheme is secure, $\mathcal{A}$ cannot succeed. This in turn means that the only way $\mathcal{A}$ can succeed in breaking Π' is to actually break the key exchange protocol Π. This is also a negligible event, since Π is secure. Taken together, these say that Π' is secure.

It is possible to give a formal proof using "transivitiy" arguments like the one given in Theorem 10.13.

Problem 9.2: The key to this attack is that Eve will masquerade as Bob to Alice, and masquerade as Alice to Bob. So, when Alice sends her public value $A = g^a$, Eve will receive it and send the public value $E = g^e$ back to Alice. Eve also sends E to Bob and receives back $B = g^b$. The point is that Alice thinks that E came from Bob and Bob thinks that E came from Alice. Now Alice and Eve have arrived at the shared secret $A^e = E^a$, resulting in key k_1, and Eve and Bob have arrived at the shared secret $E^b = B^e$, resulting in key $k2$. When Alice sends a message to Bob, Eve can decrypt the message with k_1, save it for her own purpose, and re-encrypt it to Bob with k_2.

The use of secret questions won't help, since Eve will simply pass the question on to Bob, and receive the answer to pass back to Alice, without either party realizing what is going on!

Problem 9.3: By defintion we have

$$w \oplus t = u \oplus r \oplus t = s \oplus t \oplus r \oplus t = s \oplus r = k \oplus r \oplus r = k$$

so Alice and Bob arrive at the same key.

This key exchange algorithm is insecure. The transcript for the key exchange consists of (s, u, w), which the eavesdropper has access to. Now the eavesdropper computes

$$s \oplus u \oplus w = (k \oplus r) \oplus u \oplus (u \oplus r) = k$$

and has the secret key.

Modern Cryptography
600.442
Homework 8 Solutions

November 18, 2013

Problem 10.1: The point of this exercise is that the set of all possible encryptions of 0, and the set of all possible encryptions of 1, and the set of all possible secret keys corresponding to the public key `pk` are all finite and computable by the adversary. So a computationally unbounded adversary can make all possible encryptions of plaintexts and decrypt will all possible secret keys until he finds the correct key `sk`. With `sk` in hand, the adversary can now read any message.

Problem 10.2: Since the encryption algorithm is deterministic, upon receiving a ciphertext c the adversary just needs to compute $\mathrm{Enc}_{\mathtt{pk}}(m)$ for every message m in $\mathcal{L}$, stopping when he gets to c. The time to do this is linear in $\mathcal{L}$.

Problem 10.4: Unfortunately the book is a little vague on what constitutes a key-exchange algorithm. This problem requires what we might term a "one-round" key-exchange algorithm, where Alice sends information A to Bob, Bob sends information B to Bob, and then they are able to decide on a key using their corresponding private information. (One can at least imagine key-exchange protocols which require multiple rounds of communication, perhaps with information sent in later rounds depending on the information sent in previous rounds.) If a denotes Alice's private information and b denotes Bob's private information, then there is a function f such that $f(A, b) = f(B, a) = k$, the shared key.

To turn this into a public-key encryption algorithm, Alice simply selects her public information A once and for all and makes it her public key. If Bob wants to encrypt a message to Alice, he generates his information (B, b), computes $k = f(A, b)$, and encrypts his message m using a secure private-key encryption algorithm with key k. If $c \leftarrow \mathrm{Enc}_k(m)$, then the ciphertext is (B, c). Given such a ciphertext, Alice derives the same key k as $f(B, a)$ and decrypts $m = \mathrm{Dec}_k(c)$.

An adversary $\mathcal{A}$ trying to break this public-key scheme is given the transcript (A, B) for the key exchange, as well as the ciphertext $\mathrm{Enc}_k(m_b)$ where $k = f(A, b) = f(B, a)$. Since the key exchange is secure, $\mathcal{A}$ can only recover k

from (A, B) with negligible advantage. Without the key, $\mathcal{A}$ can only guess b with negligible advantage since Π is a secure private-key encryption scheme. Together, these say that $\mathcal{A}$ has only a negligible advantage in breaking the public-key scheme.

Problem10.7: The key point of this exercise is not the individual numbers 0.01 and 0.99, but rather the fact that they are both constant values. Since $\mathcal{A}$ fails with probability 0.99, the probability that $\mathcal{A}$ will fail on k independent instances of the RSA problem is 0.99^k. The smallest value of k for which this probability is less than 0.01 is $k = 459$.

If $\mathcal{A}'$ is an adversary trying to solve the RSA problem and is given an output y, then $\mathcal{A}'$ can choose $r \leftarrow \mathbb{Z}_N^*$ and call $\mathcal{A}$ as a subroutine with input $yr^e = (xr)^e \pmod{N}$. The point is that since r is random, $\mathcal{A}'$ is giving $\mathcal{A}$ a new random instance of the problem each time. In addition, when $\mathcal{A}$ returns a value z, $\mathcal{A}'$ can compute $x' = zr^{-1} \pmod{N}$ and check if $x' = x$. If $\mathcal{A}'$ does this 459 times, he will succeed with probability 0.99.

For any probabilities $p_1 < p_2$, there is a value of k for which $(1 - p_1)^k < 1 - p_2$. This means that an adversary who succeeds with probability p_1 can be turned into an adversary who succeeds with probability p_2 with *constant* overhead, provided that it is possible to generate k random instances of the original problem.

Problem 10.11: (a) Note that if $b = 0$ then $c_2 = c_1^x$ and if $b = 1$, then $c_2 = c_1^x$ with negligible probability. (Since z is chosen at random when $b = 1$, there is a small probability that $z = xy$ which would cause a decryption failure.) Knowing x, to decrypt we simply set $b = 0$ if $c_2 = c_1^x$. Note that this is an explicit example of a public-key system where we need to allow decryption to fail with negligible probability.

(b) Let $\mathcal{A}$ be an adversary who can break this scheme with advantage $\epsilon(n)$. Since Π encrypts 1-bit messages the challenge ciphertext for the indistinguishability experiment is the encryption of a single bit b. We construct an adversary $\mathcal{A}'$ which breaks the DDH problem as follows.

- On input (G, q, g^x, g^y, h), $\mathcal{A}'$ instantiates Π with public key g and private key g^x.
- $\mathcal{A}'$ returns the challenge ciphertext (g^y, h) to $\mathcal{A}'$.
- $\mathcal{A}'$ returns the *complement* of the bit returned by $\mathcal{A}$.

The probability that $\mathcal{A}'$ returns 1, given that $h = g^{xy}$, is the probability that $\mathcal{A}$ returns 0, given that $h = g^{xy}$, which is at least $\frac{1}{2} + \epsilon(n)/2$. The probability that $\mathcal{A}'$ returns 1, given that $h \neq g^{xy}$, is at most $\frac{1}{2} - \epsilon(n)/2$. Hence

$$|\Pr[\mathcal{A}'(h = g^{xy}) = 1] - \Pr[\mathcal{A}'(h \neq g^{xy}) = 1| \geq \epsilon(n).$$

This shows that $\epsilon(n)$ is negligible.

Problem 10.14: For the chosen ciphertext attack, the adversary $\mathcal{A}$ will create two messages m_0 and m_1, each of which has its high bit set to 0. Upon receiving the challenge ciphertext c, $\mathcal{A}$ requests the decryption of the ciphertext $c' = 2^e c$. Note that c' is the encryption of twice the padded message m_b, which is simply shifting the entire padded message one bit to the left.

Since the high bit of m_b is 0, the shifted message is still a valid message. However, if the high bit of the random value r is a 1, then the padding will be ill-formed and the decryption oracle will return $\perp$. In this case, $\mathcal{A}$ takes a random guess for b. On the other hand, if the high bit of r is 0, then the padding is correctly formed and the decryption oracle will return the message. In this case, $\mathcal{A}$ easily gets b correct. The probability of success for $\mathcal{A}$ is therefore 3/4.

The reason this is effective is because the padding scheme allows for the shift of a padded message to still be a correctly padded message sometimes. This is not true of PKCS #1 v.1.5.