Question 1

(a) Give two examples where even the use of strong encryption algorithms can result in significant vulnerabilities?

1. A reflection attack. A secret key is required for encryption. If the same secret key is used in both directions of communication, an attacker is able to create a channel of communication that is authenticated. This is accomplished by the attacker using the targets own challenge as a means to authenticate the connection. This is done by forming two connections with the target, using the challenge they sent on channel one to authenticate on channel two (since the challenge is bidirectional).
2. Person in the middle attack of a Diffie-Hellman key exchange. Trudy can sit in the middle of a key exchanges between Alice and Bob. When Alice sends her key to Bob, Trudy can instead insert her key. Likewise, when Bob responds Trudy inserts her key. Now Trudy knows both the agreed upon keys. She can sit between the communication between Alice and Bob, reading their messages and even modifying them.

(b) A one-time pad must never repeat. Yet, in a series of fixed-length numbers, for example 8-bit bytes in RC4, some number must ultimately repeat, because there are only 256 8-bit numbers (0-255). Why are these two statements not contradictory?

A one-time pad is a pseudorandom stream of bits or called a keystream. The keystream must be the same length as the message to be encrypted as a $\oplus$ operation is performed on the plaintext and the keystream. Given the keystream is a pseudorandom stream of 8-bit numbers, numbers will repeat as there are only 256 choices. This is fine and not a security threat. What is a security threat is **reusing the keystream** to encrypt multiple messages.

One scenario is an attacker knows your plaintext message and intercepts the ciphertext. The attacker can easily recover the keystream and use it to decrypt future messages. This can be accomplished using the fact that $x \oplus x = 0$ and $0 \oplus y = y$.

$$m_n \oplus c_n$$
$$m_n \oplus (m_n \oplus KEY)$$
$$0 \oplus KEY$$
$$KEY$$

Alternatively, an attacker can perform a statistical analysis on a large group of intercepted ciphertext that was generated with a fixed key. Eventually figuring out the key which can be used to decrypt future intercepted ciphertext.

(c) Suppose a router between Alice and Bob can intercept Alice's messages sent to Bob and insert it's own messages. Which one of the encryption methods, a block cipher (e.g., AES) or a stream cipher (e.g., RC4), should Alice use if she suspects the adversarial router to know the plaintext corresponding to some of the ciphertext she

sends to Bob? Explain your answer.

Alice should use a block cipher encryption method. A stream cipher such as RC4 uses a one-time pad and a $\oplus$ operation to encrypt a message m. Suppose Alice wants to send a messages m_A to Bob. Trudy, knowing the contents of m_A, can replace this with her own message m_T. RC4 encrypts Alice's message into cipher text as such $c_A = m_A \oplus KEY$. All Trudy has to do is $\oplus$ her message m_T with m_A and place the result, c_T, where m_A was.

$$c_T = (m_T \oplus m_A) \oplus (KEY \oplus m_A)$$

We know $\oplus$ is associative and $x \oplus x = 0$ as well as $0 \oplus y = y$.

$$c_T = m_T \oplus KEY \oplus m_A \oplus m_A$$
$$c_T = m_T \oplus KEY \oplus 0$$
$$c_T = m_T \oplus KEY$$

Trudy has encrypted her plaintext message, without even knowing the one-time pad, and Bob will decrypt it as normal.

Question 2

(a) With regular CBC, if one ciphertext block is lost, how many plaintext blocks are lost?

When performing regular cipher block chaining (CBC), if one ciphertext block is lost then **two** plaintext blocks will be lost.

Consider c_n getting lost. During decryption, m_n is produced by decrypting c_n and $\oplus$ the result with c_{n-1}. As such, if c_n is lost then m_n will not be recoverable. Additionally, m_{n+1} is produced by decrypting c_{n+1} and $\oplus$ the result with c_n. So m_{n+1} is lost as well. But, since m_{n+1} or c_n. is not required to recover m_{n+2}, the rest of the message will be recoverable.

(b) With NCBC, why do things get back in sync if c_n and c_{n+1} are switched?

To decrypt a message block we perform the following operation where D is the decryption algorithm

$$m_n = D(c_n) \oplus c_{n-1} \oplus m_{n-1}$$

We will see switching c_n and c_{n+1} will have no effect on messages m_{n+2} and forward. First consider what m_{n+2} is composed of without switching c_n and c_{n+1}.

$$m_{n+2} = D(c_{n+2}) \oplus c_{n+1} \oplus m_{n+1}$$

We can expand m_{n+1}

$$m_{n+2} = D(c_{n+2}) \oplus c_{n+1} \oplus D(c_{n+1}) \oplus c_n \oplus m_n$$

and finally expand m_n

$$m_{n+2} = D(c_{n+2}) \oplus c_{n+1} \oplus D(c_{n+1}) \oplus c_n \oplus D(c_n) \oplus c_{n-1} \oplus m_{n-1}$$

We know that all c_i and m_i are valid for $i < n$. We also know that $\oplus$ is a communicative operation. Thus, switching c_n and c_{n+1} will have **no result** on decrypting m_{n+2} and forward.

(c) How about if a ciphertext block is lost?

If ciphertext block c_n is lost, then we will be unable to decrypt ciphertext blocks c_i for all $i \geq n$. That is, **message block n and all following message blocks will be lost.**

This is a result of requiring $D(c_n)$ to decrypt m_n. If we can't recover m_n, we can't recover m_{n+1} as $m_{n+1} = D(c_{n+1}) \oplus c_n \oplus m_n$ (it requires m_n to decrypt).

(d) How about if ciphertext block n is switched with ciphertext block $n+2$?

In this case, we will still be able to get back in sync for m_{n+3} and forward. Following the same logic as in part **(b)**, we can expand m_{n+3} out as such

$$m_{n+3} = D(c_{n+3}) \oplus c_{n+2} \oplus D(c_{n+2}) \oplus c_{n+1} \oplus D(c_{n+1}) \oplus c_n \oplus D(c_n) \oplus c_{n-1} \oplus m_{n-1}$$

As $\oplus$ is communicative, we can observe that swapping c_n and c_{n+2} has no effect on m_{n+3} and forward.

(e) How about any permutation of the first n blocks?

Following the logic laid out in parts **(b)** and **(d)**, we can see that this won't affect message blocks m_{n+1} and forward. We can recursively expand a message block m_i for $i > n$ and see that since $\oplus$ has a commutative property, the message will be decrypted properly.

Question 3

(a) Assume a good 256-bit message digest function. Assume there is a particular value, d, for the message digest and you would like to find a message that has a message digest of d. Given that there are many more 2000-bit messages that map to a particular 256-bit message digest than 1000-bit messages, would you theoretically have to test fewer 2000-bit messages to find one that has a message digest of d than if you were to test 1000-bit messages? Explain briefly.

Yes, in theory if you knew there are many more 2000-bit messages that hash to d than 1000-bit messages, you should search over the 2000-bit messages. Assuming a good message digest function, regardless of the input each bit in the output should be on about half the time. The search space for a collision isn't determined by the size of the input, but the size of the message digest. According to to famous *Birthday Problem*, we would have to search over 2^{128} messages to find two that have the same digest.

(b) Find the approximate number of messages (n) that need to be tried before finding two that had the same message digest (size k) with probability 0.8. You need to find n as a function of k. What is n when $k = 2^{256}$?

From *The Birthday Problem* in the text we know that given a message digest of size k, there is a $1/k$ probability of two inputs producing the same output. Finding the approximate number of messages n required to try to have a probability of 0.8 of two digests matching

$$\frac{1}{k} * n = 0.8$$

$$\frac{1}{k} * \frac{4k}{5} = 0.8$$

We will need to try $4k/5$ messages to have a 0.8 chance of finding two that share a message digest.

When $k = 2^{256}$, we will have to try a very large number of messages to find a collision

$$\frac{4 * 2^{256}}{5} \approx 9.263 * 10^{76} \ messages$$

(c) Consider a simple authentication protocol using hash functions where Bob sends a random number r_A to Alice and Alice responds with SHA512(K_{AB} AND r_A). Could Trudy, impersonating Bob, send Alice a sequence of challenges r_A that will reveal K_{AB}?

Trudy can pick any message r_A and send it to Alice. Alice will respond with SHA512(K_{AB} AND r_A). Now Trudy knows the hash of K_{AB} and a number she knows concatenated. She can now on her machine start to brute force guess K_{AB}. This can be done by taking each guess for K_{AB}, concatenating it with her chosen r_A, hashing the result with SHA512, and checking it against the hash from Alice. If K_{AB} is of reasonable size like 32-bits, Trudy will be able to discover it. Finally, she can now pose as Bob and communicate with Alice.

Question 4

(1) Contrast IP forwarding model with Content Centric Networking (CCN) model.

The internet has evolved greatly in the last 50 years in the type and volume of content being delivered by the underlaying architecture (IP forwarding model) has remained largely unchanged. The IP model has a focus on *where* a host is that has a desired resource. The Content Centric Networking (CCN) model shifts the focus to the *what*, using named content has its central abstraction. Using IP's engineering principles that provide robustness, CCN replaces the host a packet address names with the content.

(2) How does CCN prevent a distributed attack where an attacker requests the same content name from many different networks?

If a malicious user sends out a spam of identical *interest* packets to multiple networks, a provider cannot experience a DDoS attack and become overloaded. Regardless of the number of found matching *Data*, at each aggregation point only one *Data* packet is allowed to be forwarded to the content consumer.

(3) How does a CCN publisher control where its content travels?

All content is available to anyone on the network. The caveat is all content on the network is encrypted. CCN content publishers control who they grant decryption keys to access the encrypted content. This runs in tandem with the philosophy of CCN: remove the focus of authenticating hosts (*where*) and focus on the content (*what*).

(4) Could CCN prevent spamming (e.g., email spamming)? Explain briefly.

CCN has a number of mechanisms in place to prevent unwanted traffic at arriving to a user's machine. A key one for e-mail span is: only the number of *Data* packets requested by downstream *Interests* will be forwarded to a user. As such, as a user you could broadcast an interest in important e-mail (a white list of colleagues, newsletters you enjoy, bills, etc) and not accept anything not on this whitelist.