COM-301 Computer Security
Exercise sheet: Applied Cryptography

1. Assume an OTP-like encryption with a short key of 128 bit. This key is then being used periodically to encrypt large volumes of data. Describe how an attack works that breaks this scheme.

   Solution:
   This is an instance of double use of OTP. Each 128 bit of message is encrypted with OTP with the same key. This means that the adversary can have access to many pairs of (message XOR key) where key is the same.

   Once these pairs are available, the adversary can use any of the techniques mentioned in the class (frequency analysis, ASCII vulnerabilities,...) to gain information about the content of the message.

2. In CTR, do you need a new Initialization Vector for every message?

   Solution:
   Yes. In CTR mode, the string that is XORed with the plaintext is the output of the block cipher that takes as input IV for a given key.

   If the IV is repeated, given that the key is not changed (by definition), then the string XORed with the plaintext is also repeated.

   Thus, if two messages are encrypted using the same IV and key, their blocks will be XORed with the same string of bits. This is effectively a reuse of a one time pad, and suffers from the same problems (it is possible to learn information about the content of the message).

3. Alice wants to have confidentiality for messages in her system. Which primitive does she need to design and implement?

   Solution:
   The necessary primitive is encryption, but Alice SHOULD NOT design or implement this by herself. She should find a well-established cryptographic library. Cryptographic code is extremely error-prone: new code almost always introduces new bugs, while existing libraries have benefited from thousands of hours of expert review and testing. Even a single overlooked edge case can be catastrophic for security. Therefore, Alice should rely on a well-established cryptographic library.

4. What can you say about the:

   - Propagation of bit errors
   - Parallelizability

   for CTR and CBC modes

   Solution:

   - Errors
     - In CBC, the block that contains the error, that full block, as well as a bit in the next one get corrupted.

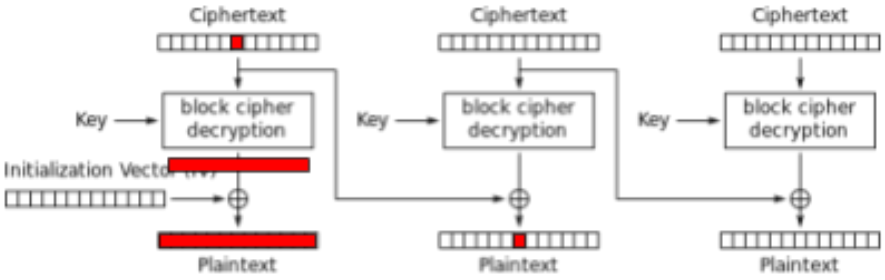

Figure 1: CBC mode decryption.

       Take a look at the decryption process. If a bit in the ciphertext is corrupted, in the corresponding block the diffusion of the block cipher creates a completely corrupted stream that when XORed with the input results in a fully corrupted block.
       For the next block, the ciphertext is correct, thus the output of the block cipher is correct. However, in the XOR step, the corrupted bit of the previous block ciphertext corrupts the bit of the plaintext.
     - In CTR, on the contrary, only the plaintext bit where the ciphertext had an error gets corrupted.
       Take a look at the decryption process (remember that for counter mode, you only need the block cipher encryption!). Here, the block cipher is used to produce a random string using the IV as input. Thus, the corrupted bit is not spread by the block cipher diffusion. It only enters into play in the XOR operations, and as such it only affects one bit of the plaintext. Also, because in counter mode there is no chaining, the error is not propagated to other blocks.
   - Parallelizability

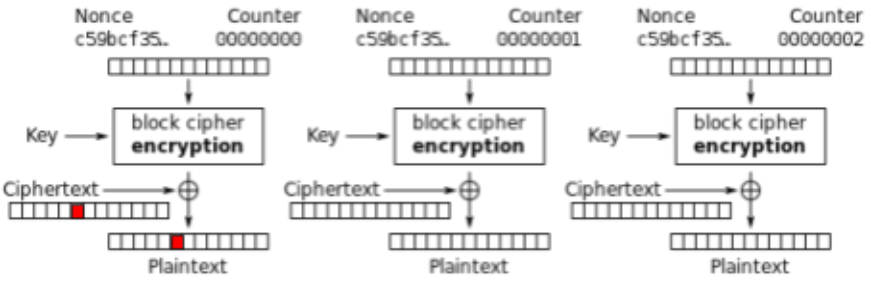

Figure 2: CTR mode decryption.

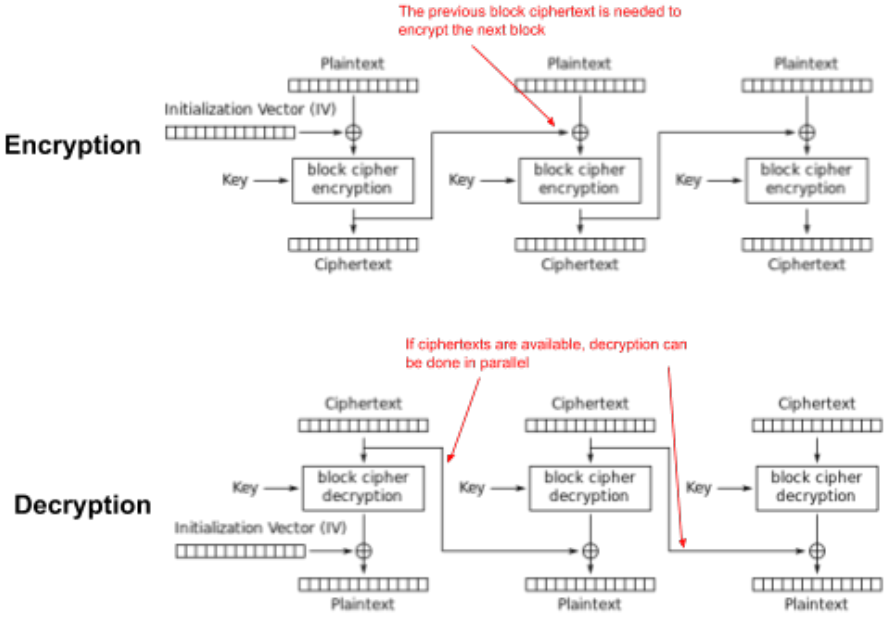

Figure 3: Encryption and Decryption in CBC mode.

– CBC is not parallelizable in encryption but it is in decryption. Let's take a look again at the processes:
In encryption, CBC requires the previous ciphertext block to create the string to be XORed. Therefore, it cannot be parallelized. In decryption, however, if one has more than one ciphertext block, the operation can be parallelized.
– In CTR mode, once the IV is available, one can produce all of the strings to be XORed with ciphertext (just increase the counter and use the block cipher encryption). Because these strings are only used in one a block, the process can be fully parallelized.

5. Which symmetric encryption is suitable for following scenarios? Justify.

   (a) A TV station wants to encrypt the premium live channels.

   (b) A message that can tolerate errors in a few bits is sent through a noisy channel.

   (c) A video channel wants to let users start watching from any arbitrary moment.

   (d) The sender wants to encrypt a photo.

   Solution:
   In green, the answer we were thinking about when writing the question. In blue, valid alternatives heard in class (if yours is not there and you want to check use email/forum). In red, wrong answers.

   (a) Stream cipher: The channel wants to send a stream of bits in realtime, and it does not know about the length of the message or next second's bits. Waiting to produce a full block before encrypting and sending the data which creates a delay. The channel requires a stream cipher to account for long streams of unknown length.
   Block cipher in CTR mode: one could also use a block cipher in counter mode. It is a bit slower (needs to wait for full blocks before decryption) but it would do the job.

   (b) In some block cipher modes, such as CBC, a one-bit error would corrupt the whole block (which is large) or even other blocks in the message. We need a cipher (or a mode) where errors are not propagated. Thus, either a stream cipher, or a block cipher in CTR mode.

   (c) To be able to skip parts of the movie or start playing the video from an arbitrary time, the decryption should have random access [be parallelizable]. The system should be able to decrypt a block without knowing the decryption of the previous block, as opposed to ciphers which always need to start decryption from the first bit. A block cipher either in CBC or CTR.

   (d) A photo is a large chunk of data with known size. A secure block cipher mode, either a block cipher in CBC or CTR would be ok. [ECB is not secure. Refer to the photo in the slides].
   A Stream cipher is not the best option as, given that we know the length, it provides no advantage with respect to the block cipher modes, and the disadvantages (low diffusion) make it suboptimal. (even though it is not the best choice, given a good justification we would give this answer as correct)

   In none of these cases (or any other case in real life!!!), would one use a One Time Pad. Even though it provides perfect secrecy, OTP requires a key which is as long as the message. Thus, it is impractical to use in any of the above-mentioned scenarios, as both generating and sharing this long key is a problem.

6. A cryptographic hash function h takes as input a message of arbitrary length and produces as output a hash value of fixed length, for example 160 bits. Certain properties should be however satisfied:

   (a) Given a message m, the hash value $h(m)$ can be calculated very quickly.

   (b) Given a hash value y, it is computationally infeasible to find an m with $h(m) = y$ (in other words, h is a one-way, or first pre-image resistant function).

   (c) It is computationally infeasible to find messages m_1 and m_2 with $h(m_1) = h(m_2)$ (remember, the function h is said to be collision-free).

   Let n be a large integer. Let $h(m) = m \bmod n$ be regarded as an integer between 0 and $n - 1$. Argue that h satisfies (a) but not (b) and (c).

   Solution:
   This function clearly satisfies (a). Given a message m, the digest $h(m) = m \bmod n$ can be computed efficiently

   However, (b) and (c) fail.

   (b) Given a message hash y (from message m). Let us take as message $m = y$. Then $h(m) = m \bmod n = y$. In other words it is possible to find m such that $h(m) = y$. Thus, h is not one-way.

   Note that if we assume that the original message is not y (e.g., $2y$), then the fact that $m = y$ is a first pre-image means that this function is also not second pre-image resistant.

   (c) Similarly to the previous answer, to show that collisions can be easily found choose any two values m_1 and m_2 that are congruent mod n. Then, $h(m_1) = h(m_2)$, so h is not collision-free

7. Bob and Alice share a secret key and use a Message Authentication Code (MAC) to authenticate messages. Bob sends a message to Alice along with its MAC. Later, Bob denies having sent the message.

   (a) Can Alice prove to a third party that Bob sent the message? Why or why not?

   (b) Would using a digital signature instead of a MAC solve this problem? Explain.

   Solution:

   (a) No, Alice cannot prove to a third party that Bob sent the message. Since both Alice and Bob share the same secret key, either of them could have generated the MAC. This means Alice has no cryptographic evidence that Bob (and not herself) created the message.

(b) Yes, using a digital signature would solve the problem. With digital signatures, only Bob possesses the private key used to sign the message. Alice (and any third party) can verify the signature using Bob's public key. This provides non-repudiation: Bob cannot deny having signed the message, because no one else could have generated a valid signature with his private key.