Problem 1): We know that the expression

$$\Pr\left[M = m \,|\, C = c\right] = \Pr\left[M = m\right], \tag{1.1}$$

which holds for some encryption scheme `(Gen, Enc, Dec)`. We apply Bayes' formula to the left-hand-side of expression 1.1 to yield

$$\Pr\left[M = m \,|\, C = c\right] = \frac{\Pr\left[M = m, C = c\right]}{\Pr\left[C = c\right]} \tag{1.2}$$

Since the $\Pr\left[M = m, C = c\right]$ term in expression 1.2 represents a Joint Probability Distribution, we also have

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

This allows the result in expression 1.2 to be equivalently expressed as

$$\Pr\left[M = m \,|\, C = c\right] = \frac{\Pr\left[M = m\right]\,\Pr\left[C = c \,|\, M = m\right]}{\Pr\left[C = c\right]}$$

which is divided by $\Pr\left[M = m\right]$ to give

$$\frac{\Pr\left[M = m \,|\, C = c\right]}{\Pr\left[M = m\right]} = \frac{\Pr\left[C = c \,|\, M = m\right]}{\Pr\left[C = c\right]} \tag{1.3}$$

By noting that expression 1.1 implies

$$\frac{\Pr\left[M = m \,|\, C = c\right]}{\Pr\left[M = m\right]} = 1,$$

the result in expression 2.2 becomes

$$\frac{\Pr\left[C = c \,|\, M = m\right]}{\Pr\left[C = c\right]} = 1$$

Multiplying this result by $\Pr\left[C = c\right]$ gives

$$\Pr\left[C = c \,|\, M = m\right] = \Pr\left[C = c\right], \tag{1.4}$$

thereby proving that $\Pr\left[M = m \,|\, C = c\right] = \Pr\left[M = m\right]$ implies $\Pr\left[C = c \,|\, M = m\right] = \Pr\left[C = c\right]$.

□

Problem 2): For a message of length $l \in \mathbb{Z}^+$, a one-time pad will be associated with three separate set spaces and three different algorithms.

The set spaces of a one-time pad are the key space $\mathcal{K}$, the message space $\mathcal{M}$, and the cipher-text space $\mathcal{C}$. Additionally these spaces are such that

$$\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^l, \tag{2.1}$$

where $\{0,1\}^l$ is the set of all binary strings having length l. Formally, the set $\{0,1\}^l$ is defined $\{0,1\}^l \equiv \{n_1, n_2, \ldots, n_l\}$ where $\forall i \in [1,l], n_i \ni [0,1] \subset \mathbb{Z}$ (i.e. either $n_i = 0$, or $n_i = 1$ for every element in the set)[1].

A one-time pad is also associated with the algorithms

- The key-generation algorithm, denoted `Gen`
- The encryption algorithm, denoted `Enc`
- The decryption algorithm, denoted `Dec`

[1]This also holds for $\mathcal{M}$ and $\mathcal{C}$

The purpose of `Gen` is to generate a key for encrypting and decrypting our message, where ee denote this key k and say that $k \in \mathcal{K}$. We say that `Gen` works by choosing a string from $\mathcal{K}$ according to the uniform distribution. From this choice of distribution, it follows that each possible key will be chosen with probability 2^{-l}.

Before we describe `Enc` and `Dec` algorithms we must define a bit-wise **XOR** on two binary strings of equal length. Let a and b be any two binary strings such that $a, b \in \{0,1\}^l$. Additionally, let us express these strings by

$$a \equiv \{a_1, a_2, \ldots, a_l\}$$

and

$$b \equiv \{b_1, b_2, \ldots, b_l\},$$

respectively. The bit-wise **XOR** of a and b is be denoted by $a \oplus b$ and expressed as the binary string

$$a \oplus b \equiv \{a_1 \oplus b_1, a_2 \oplus b_2, \ldots, a_l \oplus b_l\}$$

The elements of this binary string, the $a_i \oplus b_i$, are binary bits and are defined $\forall i \in [1, l]$ to be the traditional bit-level **XOR** of a_i and b_i, denoted $a_i \oplus b_i$. We use the truth table

a_i	0	0	1	1
b_i	0	1	0	1
$a_i \oplus b_i$	0	1	1	0

to express the definition of the transitional bit-level **XOR** and continue to the definitions of the `Enc` and `Dec` algorithms.

The `Enc` algorithm is used to encrypt the message into cipher-text based on the chosen key. We will denote the message to be encrypted by m, the cipher-text resulting from the encryption by c, and the key by k. These will each be such that $m \in \mathcal{M}$, $c \in \mathcal{C}$, and $k \in \mathcal{K}$. Using this notation and our definition for bitwise **XOR** from above, we define

$$c := m \oplus k$$

to be the expression used by `Enc` as it encrypts the message.

Inversely from the `Enc` algorithm, the `Dec` algorithm is used to decrypt the cipher-text back into the message based on the supplied key. Similarly to `Enc`, we define

$$m := c \oplus k$$

to be the expression used by `Dec` as it decrypts the message.

To prove the security of this one-time pad, consider any arbitrary message m and any arbitrary cipher-text c, where $m \in \mathcal{M}$ and $c \in \mathbb{C}$. Now, we express the probability of finding a particular c, given a particular m by

$$\Pr\left[C = c \,|\, M = m\right] \tag{2.2}$$

Using the fact that $c = m \oplus k$, this expression may be rewritten as

$$\begin{aligned}\Pr\left[C = c \,|\, M = m\right] &= \Pr\left[M \oplus K = c \,|\, M = m\right] \\ &= \Pr\left[M \oplus K = c \,|\, M = m\right] = \Pr\left[m \oplus K = c\right]\end{aligned}$$

Next, we **XOR** the random variable term in the right-hand term of this expression by m to obtain

$$\Pr\left[m \oplus (m \oplus K) = m \oplus c\right] = \Pr\left[K = m \oplus c\right],$$

because $a \oplus a = 0$ for any binary string a. Above, we defined the probability of choosing any k to be $\Pr[K = k] = 2^{-l}$. This allows us to finally obtain the relations

$$\begin{aligned}\Pr[C = c \,|\, M = m] &= \Pr[M \oplus K = c \,|\, M = m] \\ &= \Pr[m \oplus K = c] \\ &= \Pr[m \oplus (m \oplus K) = m \oplus c] \\ &= \Pr[K = m \oplus c] = \frac{1}{2^l} \qquad (2.3)\end{aligned}$$

Since or choice of m in expression 2.2 was arbitrary, the result in expression 2.3 must hold for any $m \in \mathcal{M}$. This implies that, for any $m_0, m_1 \in \mathcal{M}$, we relation

$$\Pr[K = m_0 \oplus c] = \frac{1}{2^l} = \Pr[K = m_1 \oplus c]$$

holds. This satisfies *Lemma 2.3* from the text, thus the one-time pad is perfectly secure.

Problem 3): One-time pads are difficult to use in practice because the any key must be the same length as the message and each key can be used only once.

Problem 4): Begin by defining $\prod$ to be the encryption scheme $\prod = (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ where

- The security parameter $n \in \mathbb{Z}$ and Gen are used to generate the key k by running $\texttt{Gen}(1^n) = k$
- The key k, message m, and Enc are used to produce cipher-text c by running $\texttt{Enc}_k(m) = c$
- The key k, cipher-text c, and Dec are used to recover the message m by running $\texttt{Dec}_k(c) = m$

Additionally, let m_0, m_1 be messages of the same length and c be the cipher-text generated from one of the messages by running $\texttt{Enc}_k(m_b) = c$, where $b = \{0, 1\}$. The encryption scheme $\prod$ is considered to be **CPA** secure if the probability of a polynomial time-limited adversary $\mathcal{A}$, with access to m_0, m_1 and c, determining which message was used to compute c is equal to the sum of 1/2 and any value that is negligible on the order of n.

Now denote the experiment above as $\texttt{Priv}^{\text{CPA}}_{\mathcal{A},\prod}(n)$. Let this return 0 except when $\mathcal{A}$ is able to determine which message was used to compute c then let $\texttt{Priv}^{\text{CPA}}_{\mathcal{A},\prod}(n)$ return 1. Using this notation, our definition **CPA** security can be formally stated

$$\Pr\left[\texttt{Priv}^{\text{CPA}}_{\mathcal{A},\prod}(n) = 1\right] \leq \frac{1}{2} + \texttt{negl}\ (n) \tag{4.1}$$

where $\texttt{negl}(n)$ is a negligible function of order n.

Finally, consider the case for the experiment $\texttt{Priv}^{\text{CPA}}_{\mathcal{A},\prod}(n)$ where the messages m_0, m_1 passed to the adversary $\mathcal{A}$ are such that $m_0 = m_1$ and the result of $\texttt{Enc}_k(m_i) = c_i$ is fixed each m_i in the message space. That is to say, for any fixed k, that each $c_i \in \mathcal{C}$ is determined by the result of $\texttt{Enc}_k(m_i)$ for only one $m_i \in \mathcal{M}$. In this case the result of $\texttt{Priv}^{\text{CPA}}_{\mathcal{A},\prod}(n)$ will always be 1 because the cipher-texts $c_0 = \texttt{Enc}_k(m_0)$ and $c_1 = \texttt{Enc}_k(m_1)$ are always equal thereby allowing $\mathcal{A}$ *always* to succeed every time this is case. In this case, $\prod$ does not satisfy the definition given in expression 4.1 and is therefore not **CPA** secure. Thus, we must impose an additional requirement on **CPA** secure encryption schemes.

The problem arises from the case when $m_0 = m_1$ and the when, for each fixed $m_i \in \mathcal{M}$, the function $\texttt{Enc}_k(m_i)$ always returns the the same c_i. That is to say that the operation $\texttt{Enc}_k(m_i)$ on each $m_i \in \mathcal{M}$ always determines single, unique corresponding $c_i \in \mathcal{C}$. With this in mind, we refine our definition of **CPA** security to also include the requirement that, given a fixed key k, the $\texttt{Dec}$ algorithm be non-deterministic on $m \in \mathcal{M}$. This is equivalent to requiring that the $\texttt{Dec}$ algorithm be such that any passed $m_i \in \mathcal{M}$ can return any $c \in \mathcal{C}$ with some non-zero probability, thereby making $\texttt{Dec}$ probabilistic instead of deterministic.

Problem 5): The **ECB** mode of operation is defined, it terms of the expression for the resulting cipher-text c, according to

$$c = \left\{F_k\left(m_1\right), F_k\left(m_2\right), \ldots, F_k\left(m_i\right), \ldots, F_k\left(m_l\right)\right\} \tag{5.1}$$

where $F_k\left(m_i\right)$ is a pseudo random permutation function with key k and the m_i are the blocks of the message. Since each block is directly encrypted by $F_k\left(m_i\right)$, any $m_i \in \mathcal{M}$ can result in only one unique $c_i \in \mathcal{C}$ when passed to $F_k\left(m_i\right)$ this mode is deterministic and therefore *not* **CPA** secure. Since this mode is not **CPA** secure, it *cannot* be *CCA* secure. This is follows from the fact that *CCA* security of an encryption scheme $\prod$ implies the *CPA* security of $\prod$.

We also define the **CTR** mode of operation in terms of the expression for resulting cipher-text c. For this mode of operation we have

$$c = \left\{c_0, c_1, c_2, \ldots, c_i, \ldots, c_l\right\} \tag{5.2}$$

with $c_0 = \texttt{ctr}$ and the remaining c_i defined as $c_i = r_i \oplus m_i$. Here the m_i are the blocks of the message and the r_i are defined, in terms of their index i, some random initial counter value $\texttt{ctr}$, and the keyed pseudo random permutation function $F_k\left(r_i\right)$, according to

$$r_i = F_k\left(\texttt{ctr} + i\right) \tag{5.3}$$

Since $r_i = F_k\left(\texttt{ctr} + i\right)$ and $\texttt{ctr}$ is chosen at random, the set of all r_i, $r = \left\{r_1, r_2, \ldots, r_i, \ldots, r_l\right\}$ represents a pseudo random sequence with the same length as the message. This implies that result $c_i = r_i \oplus m_i$ for each block of the message m_i depends on both on m_i and r_i instead of only m_i and the keyed pseudo random permutation function $F\left(m_i\right)$. By extension, any arbitrary message $m \in \mathcal{M}$ can be encrypted into any cipher-text $c \in \mathcal{C}$ with some non-zero probability, thereby making the **CTR** mode of operation probabilistic and thus **CPA** secure. Since the first block of the message, c_0, holds the value of $\texttt{ctr}$ in the clear, the cipher-text resulting from this mode of operation is deterministic on the value of $\texttt{ctr}$.

This enables an adversary to employ a **CCA** attack by sending $m_0 = 0^n$ and $m_1 = 1^n$ to the encryption oracle, flipping the first bit of c_1 in the cipher-text c returned by the encryption oracle to obtain c', and then sending c' to the decryption oracle to obtain either 10^{n-1} or 01^{n-1}. The possible results from the decryption oracle respectively imply that either m_0 was enciphered to into c or m_1 was enciphered into c thus giving the adversary two messages, cipher-text associated with each message, and the value of `ctr` used for encrypting both message. This information allows the adversary to eventually to recover the pseudo random permutation function F_k and its associated key used in to encrypt these messages. This attack is made possible because only the first bit in the cipher-text for m was changed and this first block of cipher-text is *directly* dependent on only the corresponding message block m_1 and the key k when the the value of `ctr` is known. The **CCA** attack exploits the fact that encryption in **CTR** mode becomes deterministic the on the values of `ctr` and some cipher-text are known.

Problem 6): Since any **CPA** secure encryption scheme requires than any message $m \in \mathcal{M}$ can be encrypted into any $c \in \mathcal{C}$ with some non-zero probability, any cipher-text c can correspond to the encryption of several different messages, thereby preventing an eavesdropper from learning anything about plain-text by comparing two cipher-texts for equivalence.

Problem 7): Each block in a cipher-text from an encryption, under **CBC** mode of operation, is dependent on either the cipher-text from the block before it or the value of `IV`, they must be encrypted or decrypted serially. Therefore, under the **CBC** mode of operation, there is no speed increase, in either encryption and decryption, available from parallel processing.

Problem 8): This **MAC** is *not* secure because it **XOR**s the 'tags' for each message block instead of **XOR**ing all of the blocks together and then generating a tag from that result. This is a problem because it does not prevent an adversary from changing the order of any of the blocks. To see this, consider any two messages $m, m^\star \in \mathcal{M}$ such that $m = \{m_1, m_2, \ldots, m_k, \ldots, m_l\}$, $m^\star = \{m_1^\star, m_2^\star, \ldots, m_k^\star, \ldots, m_l^\star\}$, $m_i = m_j^\star$, $m_j = m_i^\star$, and $m_k = m_k^\star$ for all other $k < l$. Then, clearly we have

$$t = F_k(m_1) \oplus F_k(m_2) \oplus \cdots F_k(m_i) \oplus \cdots \oplus F_k(m_j) \oplus \cdots \oplus F_k(m_l)$$

which, by the the of **XOR**, is equivalent to

$$t = F_k(m_1) \oplus F_k(m_2) \oplus \cdots F_k(m_j) \oplus \cdots \oplus F_k(m_i) \oplus \cdots \oplus F_k(m_l)$$

Applying our definitions for m and $m^\star$ from above, we clearly see that

$$\begin{aligned} t &= F_k(m_1) \oplus F_k(m_2) \oplus \cdots F_k(m_j) \oplus \cdots \oplus F_k(m_i) \oplus \cdots \oplus F_k(m_l) \\ &= F_k(m_1^\star) \oplus F_k(m_2^\star) \oplus \cdots F_k(m_i^\star) \oplus \cdots \oplus F_k\left(m_j^\star\right) \oplus \cdots \oplus F_k(m_l^\star) = t^\star \end{aligned}$$

thereby showing that an adversary can indeed change the message order with out altering the message tag.

Problem 9): We can calculate the probability of finding a collision here. There are $N_{TOT} = \prod_{i=1}^{l}\{2^n\} = = (2^n)^l$ total possible messages with l blocks and block-length n. Out of these, there are $N_{NO-COL} = = \prod_{i=1}^{l}\{2^n - i + 1\}$ messages that will have no collisions because one message is removed from the number available for each subsequent block. Therefore, the number of messages with collisions in the *must* be

$$\begin{aligned} N_{COL} = N_{TOT} - N_{NO-COL} &= \prod_{i=1}^{l}\{2^n\} - \prod_{i=1}^{l}\{2^n - i + 1\} \\ &= (2^n)^l - \prod_{i=1}^{l}\{2^n - i + 1\} \end{aligned} \tag{9.1}$$

This allows the probability of finding a collision to be given as

$$\Pr[COL] = 1 - \prod_{i=1}^{l}\{2^n - i + 1\} \tag{9.2}$$

For sufficiently large n, this yields an acceptable probability of collision, therefore this hash function would be secure within the Merkle-Damgard paradigm.

Problem 10): This scheme does not provide authentication. In any **CPA** secure scheme, it is required that multiple, different messages can be encrypted into any arbitrary cipher-text with some non-zero probability. Thus, we may have that $\texttt{Enc}_k(m_1) = \texttt{Enc}_k(m_2) = c$ even if $m_1 \neq m_2$. In this case $t_1 = t_2$ would be true. This is due to the fact that the hash function in this scheme is not keyed, therefore any arbitrary cipher-text c always yields the same tag $t =$ **hash** (c). Therefore, this scheme only ensures that the cipher-text has not been tampered with, not that it is authentic. It does nothing to ensure that the underlying message is being represented by the received cipher-text or the cipher-text itself are authentic.

Problem 11): Define the cipher-texts c and c', encrypted on a one-time pad with the same key, from messages m and m', respectively, as $c = m \oplus k$ and $c' = m' \oplus k$. Now, **XOR** these expressions together to obtain

$$c \oplus c' = (m \oplus k) \oplus (m' \oplus k)$$

Since $k \oplus k = 0$, our result simplifies to

$$c \oplus c' = m \oplus m'$$

thereby giving an adversary a relation between cipher and message texts.

Problem 12): Encryption keys can be securely reused with a stream cipher through the use of random initialization vectors and augmented pseudo random functions, $G(IV, k)$ which accept both and initialization vector and a seed and remain pseudo random even when the IV is known. Is passed at the beginning of the current session so that $\texttt{Enc}$ is defined as

$$\texttt{Enc} := \langle IV, G(k, IV) \oplus m \rangle$$

Additionally, we have

$$\texttt{Dec} := G\left(k, IV\right) \oplus c$$

as the new definition for `Dec`.

Problem 13): Storing the hash of the *username* concatenated with the *password* would not be considerably more secure than just storing the hashes of the passwords. This is due to the fact that an attacker trying to crack passwords already has a list of usernames; therefore this would not significantly increase the amount of time required to check the stolen hashes against a precomputed table of hashes. The only impediment is that the precomputed table would have to include hashes of longer strings, therefore possibly requiring more time to compute.

Problem 14): This encryption approach will accept an input message along with two n-bit keys, where $n = 128$, 256, or 512 bits. The block level encryption algorithm for this will be **AES** with n-bits. This system uses the following steps to encrypt a message

- Read the input into an array of Strings with l total Strings. This array will be denoted $m\left[\right]$.
- An overall message identifier, r_{MSG}, will be chosen at random and stored as an n-bit number.
- A value for a counter, ctr, to be used in the **CTR** mode of encryption will be randomly chosen.
- The length of the longest string in $m\left[\right]$ will be found and stored as l_{TEMP}.
- The number of blocks per message line will be computed according to $l_i = \text{ceil}\left(l_{TEMP}/n\right)$.
- For each message line, $i = 1 : l$, the initial message stored in $m\left[i\right]$ will be copied into a temporary string $tempM$. This temporary string will then be padded with random data until its length is equal to $l_i \cdot n$, thereby allowing it to be broken into l_i blocks of equal size.
- A message line identifier, r_i, will be chosen at random and stored.
- A **MAC**-tag will be computed for this line by computing the **CBC-MAC** of $\left(r_i, i, \left|m_i\right|, m_i\right)$ using $k_3 = r_i \oplus k_2$. Here, r_i, i, l, and $\left|m_i\right|$ will all be stored as n-bit numbers, just as r_{MSG} was. The value of the **CBC-MAC** for a line i will be stored as t_i.

- Each message line will have $l_i + 4$ blocks of cipher-text, with each block being n-bits long.
  - The first extra block, to be added to the beginning of the cipher-text for the line will be the encryption of of r_i for that line of the message.
  - The second extra block, will be added immediately after the first extra block will contain the encryption of the n-bit value for i.
  - The third extra block, as previously, will be added immediately after the second and will contain the encrypted n-bit value for $|m_i|$.
  - The fourth extra block will be added to the end of the cipher-text for the line will be the encryption of the t_i value for this line. These encryptions will use the same key as the key used for encrypting the message text, k_1.
- Therefore, for a given line of the message, will have the cipher text

$$c_i = \left\{\texttt{Enc}_{k_1}^{ctr}(r_i), \texttt{Enc}_{k_1}^{ctr}(i), \texttt{Enc}_{k_1}^{ctr}(|m_i|), \textbf{ for } j = 1 : l_i \left[\texttt{Enc}_{k_1}^{ctr}(m_j)\right], \texttt{Enc}_{k_1}^{ctr}(t_i)\right\}$$

- The cipher-text for the entire message will be completed by running this for all message lines $i = 1 : l$ (resulting in cipher-text lines c_i for $i = 1 : l$), followed by pre-pending the initial value for ctr (in the clear), the encrypted value of r_{MSG}, the encrypted value of l_i, and the encrypted value of l.
- Next, a **MAC**-tag for the entire message will be computed using $\texttt{SHA-256}_{k_2}(ctr, r_{MSG}, l_i, l, m)$, where m is the unaltered message (i.e. no r_i, t_i, etc.). This **MAC**-tag will be stored as t_{MSG}, then be encrypted by $\texttt{Enc}_{k_1}^{ctr}(t_{MSG})$. The result of this encryption will be appended to the end of the complete cipher-text described above.
- This means that the cipher-text of the entire message can be represented according to

$$c = \left\{ctr, \texttt{Enc}_{k_1}^{ctr}(r_{MSG}), \texttt{Enc}_{k_1}^{ctr}(l_i), \texttt{Enc}_{k_1}^{ctr}(l), \textbf{ for } i = 1 : l \; [c_i], \texttt{Enc}_{k_1}^{ctr}(t_{MSG})\right\}$$

To decrypt a message, this system will follow the following steps

- Remove the first n bit block. This will be stored as ctr. Decrypt the remaining blocks using $\texttt{Enc}_{k_1}^{ctr}$
- Recover r_{MSG} from the first decrypted block of cipher-text, l_i from the second, and l from the third.

- Use l to determine how many lines of cipher-text there are to decrypt and l_i to determine how many blocks there are per line.
- Using the values of l and l_i, along with the knowledge that each line of cipher text has four blocks that are not related to the message, reconstruct the message text (including padding), and store the r_i, i, $|m|$, and t_i values for each line.
- Use the value of $|m|$ to strip off the padding characters for each line.
- Use the restored message line along with $|m|$, i, and r_i to compute the **CBC-MAC** for that line, as described above. Compare this computed value to the one stored in t_i, if they match then proceed to the next line.
- Once the message has finally had all the padding stripped off through the last line, use all of the just decrypted information to compute t_{MSG} as described above. Compare this computed value for t_{MSG} to the one passed and decrypted from the message. If they match, the message is authentic and decryption is complete.

The features of this system, and their reasoning, are

- The is a **MAC**-tag for each line of the message - This allows messages to be broken into pieces (packets) and still be securely encrypted and authenticated with the same two keys.
- Each line is authenticated independently - With every having a tag that is independent other lines and of the entire message, providing both authentication for each line in addition to a check on the authentication of the entire message.
- Padding by line - This allows packets with differently lengths to be sent without giving an adversary any information about the size of each line. All the adversary can know is the length of the longest line.
- Random identifiers at both the message and line levels - This prevents an adversary from swapping message lines or forging the entire message.
- Counter for every line - This prevents an adversary from swapping message blocks.
- Using **CBC-MAC** for per line authentication - This provides an authentication scheme for every line of the message that is independent of both the encryption of that line (or the entire message)

and the authentication for the entire message. This is why the **CBC-MAC** authentication uses the **XOR** of the second (authentication) key and the random line identifier (r_i) as its key. This also means that message encryption (using **AES** in **CTR** mode) and full message authentication (using **SHA-256**) employ different techniques from that used for the authentication of each line.

- The use of **AES** in **CTR** mode allows for the encryption of messages in a relatively parallel fashion, the number of blocks in each line is known so the counter value for any block in any line is know before encryption starts based on the steps in this scheme and the size of the message. The only limitation will be if computing the **CBC-MAC** for each line is time-consuming. However, since the **CBC-MAC** of each line is independent of all other lines the **CBC-MAC** for multiple lines may be computed simultaneously.
- The use of **SHA-256** to compute a full message level authentication tag - This tag is computed based message level information that need not be repeated for every line. This limits the chance of a collision between the **CBC-MAC** tags for each line. If the **MAC**-tag for each line included this, it would increase the chance of collisions, however slightly, as the **CBC-MAC** tags for any two lines would be guaranteed to have at least this information in common. Additionally, this would increase the time required to compute the **CBC-MAC** for each line by increasing the amount of information being passed, since every line has a **CBC-MAC** tag, this overhead could become significant. Finally, the full message **MAC**-tag also serves to ensure that the value of ctr (which *must* be passed in the clear) has not been tampered with and is indeed genuine.

Because this system relies on keys that are of either 128, 256 or 512-bits in length, the system would be secure for sending packets on a network with a packet size of 500 bits. This is because this system would treat each packet as a ”line”, and each line is securely authenticated individually.

Problem 15): Yes, the system proposed in **Problem 14** would be able to securely handle packets of varying sizes. This is due to the fact that the algorithm pads all packets to a the multiple of n-bits required to encode the largest packet, thus an adversary will not be able to determine anything except an upper bound for the length of the largest packet.