Problem 1. Suppose we can find two message/hash pairs \( \langle M_1, H(M_1) \rangle \) and \( \langle M_2, H(M_2) \rangle \) such that \( M_1 \neq M_2 \) and \( H(M_1) = H(M_2) \), where \( H \) is the large hash function that satisfies \( H : X^{\leq L} \rightarrow X \). Then, there exist two distinct Merkle hash trees \( T_1 \) and \( T_2 \) whose outputs are identical.

We can find a collision for the compression function using a top-down side-by-side comparison of our two trees, looking across trees for a case where the outputs of \( h \) are the same, but the inputs differ. Starting at the root, suppose that either the first block or \text{msg-len} \ (\text{thus height}) \ of our trees differ, in this case, clearly we have found a collision in \( h \), so we can stop our search. Otherwise, note that \text{msg-len} \ is the same, so the trees have exactly the same height and structure. Because they have the same structure, we don’t have to worry about null values, since they will appear in the same places in both trees. We only need to look at the nodes where \( h \) is actually evaluated with two inputs.

We can continue down examining the \( i \)th node of \( T_1 \) and \( T_2 \) and compare the inputs. If they differ, then we have found a collision for the compression function, and the procedure terminates. If not, we proceed to level \( i + 1 \) and repeat the same procedure. Note that since \( M_1 \neq M_2 \), we cannot go through the entire tree without finding a collision. It follows that one can find a collision in the compression function.

Problem 2. \( f_1 \): For any given value of \( y \), is it actually possible to find a value \( x \) to force a given output \( c \):
\[
f_1(x, y) = E(y, x) \oplus y = c
\]
\[
\iff E(y, x) = c \oplus y
\]
\[
\iff x = D(y, c \oplus y)
\]
Thus, we can pick an arbitrary values \( c, y, y' \) such that \( y \neq y' \), and exhibit the following colliding pairs:
\[
\begin{align*}
& f_1 \left( D(y, c \oplus y), y \right) \\
& f_1 \left( D(y', c \oplus y'), y' \right)
\end{align*}
\]
Since \( y \neq y' \), this is a non-trivial collision.

\( f_2 \): Similarly, we can pick an arbitrary value of \( x \) and solve for \( y \) to produce a desired output \( c \):
\[
f_2(x, y) = E(x, x \oplus y) = c
\]
\[
\iff x \oplus y = D(x, c)
\]
\[
\iff y = x \oplus D(x, c)
\]
Thus, we can pick an arbitrary values \( c, x, x' \) such that \( x \neq x' \), and exhibit the following colliding pairs:
\[
\begin{align*}
& f_2 \left( x, x \oplus D(x, c) \right) \\
& f_2 \left( x', x' \oplus D(x', c) \right)
\end{align*}
\]
Since \( x \neq x' \), this is a non-trivial collision.

Problem 3. (a) Since \( A \) and all \( B_i \in B \) share the secret key \( k \), any \( B_i \) can send to the parties \( B \setminus \{B_i\} \) a message \( M \) appended with the MAC under \( k \) of \( M \). To the recipients this looks exactly like a message sent by \( A \), and hence, the recipient is not sure the message came from \( A \).
(b) \( B_i \) can successfully fool \( B_j \) \((i \neq j)\) if and only if \( B_i \) has every key that \( B_j \) has. This is easy to see since \( B_j \) verifies a message by verifying the MACs corresponding to the keys he has. Thus, for the scheme to work, it must be the case that for every \( i \neq j \), \( S_j \not\subseteq S_i \) where \( S_i, S_j \) are the sets of keys held by \( B_i \) and \( B_j \), respectively.

Note that we use the assumption of non-collusion to ensure that if some proper subset of parties \( B^* \subset B \) have between them every key, they cannot work together to fool the parties in \( B \setminus B^* \).

(c) Let the keys be \( k_1, k_2, k_3, k_4, k_5 \). We note that \( 10 = \binom{5}{3} \). Hence each \( S_i \) need only contain two keys. The subsets are:

\[
S_1: \{k_1, k_2\} \quad S_2: \{k_1, k_3\} \quad S_3: \{k_1, k_4\} \quad S_4: \{k_1, k_5\}
\]

\[
S_5: \{k_2, k_3\} \quad S_6: \{k_2, k_4\} \quad S_7: \{k_2, k_5\} \quad S_8: \{k_3, k_4\}
\]

\[
S_9: \{k_3, k_5\} \quad S_{10}: \{k_4, k_5\}
\]

Note that for any two \( S_i, S_j \) \((i \neq j)\), \( S_i \) and \( S_j \) differ in at least one element.

(d) If users \( i \) and \( j \) collude, they must have at least 3 (possibly 4) different keys between them. This means they can produce \( \binom{3}{2} = 3 \) different key pairs. Two of those pairs are \( S_i \) and \( S_j \), but there must be at least one other pair for which they own both keys. The colluding users can generate a MAC which will be accepted by the user who owns that key pair. The colluding users can trick even more users if they own 4 different keys between them.

A simple example that shows this is if users 1 and 2 from the previous answer collude. Together, they own \( k_2 \) and \( k_3 \), so they can produce a MAC which will be accepted by user 5.

**Problem 4.** When decrypting a two block ciphertext in randomized CBC mode, we get \( m = D(k, c_1) \oplus c_0 \).

Since we don’t have the key, we can’t modify \( c_1 \). Therefore, to modify the decrypted message, we change \( c_0 \) (the IV) to \( c_0 \oplus 000000^\ast bob^00..00 \oplus 000000^\ast mel^00..00 \). When the ciphertext is decrypted, we will get \( m' = D(k, c_1) \oplus c_0 \oplus bob \oplus mel = m \oplus bob \oplus mel \). This removes ”bob” from the original message and replaces it with ”mel”.

After applying this change, the new ciphertext is:

65e26545815e038c6593660ed8638532 b365828d548b3f742504e7203be41548

where the bold characters represent the bytes that were changed.

**Problem 5.** (a) This does not provide authenticated encryption. We construct an adversary for the ciphertext integrity game.

1. The adversary submits an arbitrary message \( m \) and receives the ciphertext \( c_1 = (c_1', c_1') \).
2. The adversary then queries for the encryption of another arbitrary message \( m' \neq m \) and receives the ciphertext \( c_2 = (c_2', c_2') \).
3. Finally, the adversary submits the ciphertext \( c = (c_1', c_2') \).

Since \( m \neq m' \), \( c_1' \neq c_2' \), the adversary wins the ciphertext integrity game with advantage 1.

(b) This scheme provides authenticated encryption. We show that the scheme provides both ciphertext integrity and is CPA-secure. First, we show that if \( (E, D) \) provide ciphertext integrity, then \( (E_2, D_2) \) provide ciphertext integrity. As usual, we prove the contrapositive. Suppose we have an efficient adversary \( \mathcal{A} \) that wins the ciphertext integrity game against \( (E_2, D_2) \) with non-negligible advantage. We construct an adversary \( \mathcal{B} \) that wins the ciphertext integrity game against \( (E, D) \) with the same advantage:

1. Start running \( \mathcal{A} \).
2. Whenever \( \mathcal{A} \) requests the ciphertext on a message \( m \), forward the query to the challenger to obtain \( c \leftarrow E(k, m) \).


3. At some point, $A$ outputs a new ciphertext $c_\ast = (c'_\ast, c''_\ast)$. Output $c'_\ast$.

We are perfectly simulating the view $A$ expects when interacting with a challenger for $(E_2, D_2)$, so by assumption, with non-negligible probability $\varepsilon$, $A$ will produce a new ciphertext $c_\ast$ that is valid for $(E_2, D_2)$. This means that $c'_\ast = c''_\ast$, and moreover, is distinct from any ciphertext received from the challenger. Thus $B$ achieves the same advantage $\varepsilon$ in the ciphertext integrity game.

Next, we show that $(E_2, D_2)$ is CPA-secure if $(E, D)$ is CPA-secure. Again, we proceed by contrapositive. Suppose $A$ is a CPA-adversary for $(E_2, D_2)$ that achieves distinguishing advantage $\varepsilon$. Then, we can construct a CPA-adversary $B$ for $(E, D)$ as follows:

1. Start running $A$.
2. Whenever $A$ makes a query $(m_\ell, m_r)$, forward the query to the challenger to obtain a ciphertext $c$.
   Forward $(c, c)$ to $B$.
3. Output whatever $A$ outputs.

It is easy to see that we have perfectly simulated the view of $A$ interacting with a challenger for $(E_2, D_2)$, so with the same advantage $\varepsilon$, adversary $B$ will win the CPA game against $(E, D)$, thus concluding the proof. We conclude that since $(E_2, D_2)$ provides CPA security and ciphertext integrity, it provides authenticated encryption.

(c) This scheme does not provide authenticated encryption. We construct an adversary for the ciphertext integrity game.

1. The adversary queries for the ciphertext on an arbitrary message $m$, and receives the cipher text $(c_1, c_2)$.
2. The adversary submits $(c_2, c_1)$ as his forged ciphertext.
3. With high probability (certainly non-negligible), the adversary wins the ciphertext integrity game.

With high probability, $c_1 \neq c_2$, and the adversary wins the challenge with non-negligible advantage.

(d) Does not provide authenticated encryption. This scheme does not provide CPA-security. To play the CPA game, the adversary makes no queries and plays the semantic security game.

1. The adversary selects two messages $m_0 \neq m_1$ of equal length and submits them to the challenger.
2. The challenger chooses a random $m_b$ and gives the adversary $(E(k, m_b), H(m_b))$.
3. Since $H$ is a hash function known to the adversary, the adversary computes $H(m_0)$ and $H(m_1)$.
4. If $H(m_b) = H(m_1)$, the adversary outputs 1, and otherwise 0.

Since $H$ is collision-resistant, $H(m_0) \neq H(m_1)$ (except perhaps with negligible probability). Thus, the adversary achieves advantage negligible close to 1.

Problem 6. There are multiple potential solutions to this problem which successfully produce a new, valid ciphertext. One attack is described below.

1. The adversary picks an arbitrary message $m$, computes the hash $H(m)$ and sends the message $(H(m), m)$ to the challenger.
2. The challenger encrypts using randomized CBC and returns ciphertext $(c_0, c_1, c_2, c_3)$.
3. The adversary submits the ciphertext $(c_1, c_2, c_3)$, which will be accepted as a valid ciphertext.
Here, we show why the new ciphertext is valid. From the original CBC encryption, \(c_2 = E(k, c_1 \oplus H(m))\), and \(c_3 = E(k, c_2 \oplus m)\). Because \(c_1\) is now the IV, decrypting \(c_2\) results in \(m_1 = D(k, c_2) \oplus c_1\). Plugging in the value of \(c_2\), we get \(m_1 = D(k, E(k, c_1 \oplus H(m))) \oplus c_1 = c_1 \oplus H(m) \oplus c_1 = H(m)\). When we decrypt \(c_3\), we get \(c_2 = c_2 \oplus m \oplus c_2 = m\). Therefore, the full decrypted message is \((H(m), m)\), so when \(D'\) decrypts, \(t = H(m)\), and the message will be accepted.

**Problem 7. (a)** Here’s the algorithm:

```plaintext
input : Precomputed table \(T\) where \(T[k] = g^k\) for \(0 \leq k \leq 2^w - 1\).
output: \(g^x\)
1  \(A \leftarrow 1;\)
2  for \(x_i\) from \(x_{d-1}\) to \(x_1\) do
3      \(A \leftarrow A \times T[x_i];\)
4  for \(j \leftarrow 1\) to \(w\) do
5      \(A \leftarrow A \times A;\)
6  end
7  \(A \leftarrow A \times T[x_0];\)
8  return \(A\)
```

**Algorithm 1: Left-to-right exponentiation**

Since \(x_i\) is a digit in base \(2^w\), \(g^{x_i}\) is given by \(T[x_i]\). We multiply the accumulator by \(T[x_i]\) on each iteration of the outer loop. The inner loop squares the accumulator \(w\) times. Thus the outer loop does \(w + 1\) multiplications per iteration and runs for \(d - 1\) iterations where \(d\) is the number of digits of \(x\) in base \(2^w\).

Note that \(p\) has \(1 + \lceil \log_2(p) \rceil\) digits in base \(2^w\). Therefore:

\[
d = 1 + \lceil \log_2(p) \rceil
\]

\[
d = 1 + \left\lceil \frac{\log_2 p}{\log_2 2^w} \right\rceil
\]

\[
d - 1 = \left\lfloor \frac{\log_2 p}{w} \right\rfloor
\]

Thus the loop runs for \(\frac{\log_2 p}{w}\) iterations. Multiplying by the number of multiplications per iteration we get:

\[
(w + 1) \times \frac{\log_2 p}{w} = \log_2 p + \frac{\log_2 p}{w}
\]

\[
= (1 + \frac{1}{w}) \log_2 p
\]

Full explanation of why this algorithm works.

\[
x = x_0 + x_1 \cdot 2^w + x_2 \cdot (2^w)^2 + \cdots + x_{d-1} \cdot (2^w)^{d-1}
\]

\[
g^x = g^{x_0 + x_1 \cdot 2^w + x_2 \cdot (2^w)^2 + \cdots + x_{d-1} \cdot (2^w)^{d-1}}
\]

\[
= g^{x_0} \cdot g^{x_1 \cdot 2^w} \cdot g^{x_2 \cdot (2^w)^2} \cdots g^{x_{d-1} \cdot (2^w)^{d-1}}
\]
was precomputed, we can substitute $T[x_i]$:

$$= T[x_0] \cdot (T[x_1])^{2w} \cdot (T[x_2])^{2w} \cdots (T[x_{d-1}])^{2w}$$

$$= T[x_0] \cdot (T[x_1])^{2w} \cdot (T[x_2])^{2w} \cdots (T[x_{i}])^{2w} \cdots (T[x_{d-1}])^{2w}$$

It’s not that hard to see why these are equivalent. Basically the innermost term is raised to the power of $2^w$ $d-1$ times:

This is exactly how the algorithm computes the answer, starting at the innermost term and evaluating outwards, using repeated squaring to raise each term to $2^w$ in $w$ multiplications.

(b) This algorithm to precomputes table $T$ in $2^w$ multiplications:

```plaintext
input : g and w
output: Precomputed table T where T[k] = g^k for 0 ≤ k ≤ 2^w − 1
1 T[0] ← 1;
2 for i ← 1 to 2^w − 1 do
3     T[i] ← T[i−1] \times g;
4 end
5 A ← A \times T[x_0];
6 return T
```

Algorithm 2: Precomputation

Thus precomputing the table and running the algorithm using from part a will take $2^w + \left(1 + \frac{1}{2}\right) \log_2(p)$ multiplications in the worst case.

(c) The question boils down to computing the $w$ that minimizes our answer to the previous part:

$$\arg\min_{w \in \mathbb{N}} \left(2^w + \left(1 + \frac{1}{w}\right) \times 256\right)$$

$$= 4$$

Note that the window must be an integer, so we choose $w = 4$. 

5