CPSC 418 / MATH 318 — Introduction to Cryptography
ASSIGNMENT 2

Name: Harjee Johal
Student ID: 30000668

Problem 1 — Arithmetic in the AES MixColumns operation (22 marks)

(a) i. (a) (i) For this question, we are asked to prove that in AES MIXCOLUMNS arithmetic multiplying any 4-byte vector by y is a circular left shift by one byte. Suppose that a is a 4-byte vector such that $a = (a_3, a_2, a_1, a_0)$. Let $a(y) = a_3y^3 + a_2y^2 + a_1y + a_0$ be the polynomial representation of a. Thus, $a(y) * y$ is:

$$\begin{aligned} a(y) * y &= (a_3y^3 + a_2y^2 + a_1y + a_0) * y \\ &= a_3y^4 + a_2y^3 + a_1y^2 + a_0y. \end{aligned}$$

Next, we must perform reduction of $a(y)*y$ modulo $M(y)$. We are told that $M(y) = 0$ and that $M(y) = y^4+1$. From this, can determine that since $y^4+1 = 0$, then $y^4 = 1$. Applying this to $a(y) * y$, we get:

$$\begin{aligned} a(y) * y &= a_3y^4 + a_2y^3 + a_1y^2 + a_0y \\ &= a_3(1) + a_2y^3 + a_1y^2 + a_0y \\ &= a_2y^3 + a_1y^2 + a_0y + a_3. \end{aligned}$$

From this, we can see that $a(y)*y = (a_2, a_1, a_0, a_3)$ in vector form. When we compare the vector form of $a(y) * y$, (a_2, a_1, a_0, a_3) to the vector form of $a(y)$, (a_3, a_2, a_1, a_0), we can see that the bytes of $a(y) * y$ are the same bytes of $a(y)$, except that they have been circularly shifted to the left by one. Therefore, in AES MIXCOLUMNS arithmetic, the multiplication of any 4-byte vector a will result in its bytes being shifted circularly left by one byte.

ii. For this question, we are asked to prove that in AES MIXCOLUMNS arithmetic, $y^i = y^j$ for any integer $i \geq 0$ where $j \equiv i \ (mod 4)$ with $0 \leq j \leq 3$. If $i \equiv j \bmod (4)$, and i is an integer, then i can be rewritten in the form $i = 4k + j$, where k is an integer such that $i/4 = k$. Using this assertion, we can turn the equation $y^i = y^j$ into:

$$\begin{aligned} y^i &= y^j \\ y^{4k+j} &= y^j \\ (y^4)^k y^j &= y^j. \end{aligned}$$

We are told that in AES MIXCOLUMNS arithmetic, $M(y) = y^4 + 1 = 0$. From this, we find that $y^4 = 1$. We can use this equation substitute y^4 with 1, giving us:

$$(1)^k y^j = y^j$$
$$y^j = y^j.$$

Thus, we can see that in this arithmetic, $y^i = y^j$ for any integer $i \geq 0$ where $j \equiv i \pmod 4$ with $0 \leq j \leq 3$.

iii. We are asked to prove that in this arithmetic, the multiplication of any 4-byte vector by $y^i \geq 0$ is a circular left shift by j bytes, where $j \equiv i \pmod 4$ with $0 \leq j \leq 3$. Suppose that a is a 4-byte vector represented as (a_3, a_2, a_1, a_0) .Let $a(y) = a_3y^3 + a_2y^2 + a_1y + a_0$ be the polynomial representation of a. In this arithmetic, we are told that $M(y) = y^4 + 1$ and that $M(y) = 0$. From this we get $y^4 = 1$. From part (a)(ii) we know that $y^i = y^j$ for any integer $i \geq 0$ where $j \equiv i \pmod 4$ with $0 \leq j \leq 3$. Therefore, there are four cases to be examined:

Case 1: j = 0. If $j = 0$, then $y^i = y^j = y^0 = 1$. Therefore, when we multiply $a(y)$ with y^i, we get:

$$\begin{aligned} a(y) * y^i &= a(y) * 1 \\ &= a(y). \end{aligned}$$

Therefore, $a(y) * y^0 = a(y) = (a_3, a_2, a_1, a_0)$, which is a left circular shift of a by 0 bytes. Thus, in this case the statement is proven true.

Case 2: j = 1. If $j = 1$, then $y^i = y^j = y^1 = y$. Therefore, when we multiply $a(y)$ with y^i, we get:

$$\begin{aligned} a(y) * y^i &= a(y) * y \\ &= (a_3y^3 + a_2y^2 + a_1y + a_0) * y \\ &= a_3y^4 + a_2y^3 + a_1y^2 + a_0y. \end{aligned}$$

Using the fact that in this arithmetic $y^4 = 1$, we can reduce this equation to:

$$\begin{aligned} a(y) * y &= a_3 + a_2y^3 + a_1y^2 + a_0y \\ &= a_2y^3 + a_1y^2 + a_0y + a_3. \end{aligned}$$

Therefore, $a(y) * y^1 = a(y) = (a_2, a_1, a_0, a_3)$, which is a left circular shift of a by 1 byte. Thus, in this case the statement is proven true.

Case 3: j = 2. If $j = 2$, then $y^i = y^j = y^2$. Therefore, when we multiply $a(y)$ with y^i, we get:

$$\begin{aligned} a(y) * y^i &= a(y) * y^2 \\ &= (a_3y^3 + a_2y^2 + a_1y + a_0) * y^2 \\ &= a_3y^5 + a_2y^4 + a_1y^3 + a_0y^2. \end{aligned}$$

Using the fact that in this arithmetic $y^4 = 1$, we can reduce this equation to:

$$\begin{aligned} a(y) * y &= a_3y + a_2 + a_1y^3 + a_0y^2 \\ &= a_1y^3 + a_0y^2 + a_3y + a_2. \end{aligned}$$

Therefore, $a(y) * y^2 = a(y) = (a_1, a_0, a_3, a_2)$, which is a left circular shift of a by 2 bytes. Thus, in this case the statement is proven true.

Case 4: j = 3. If $j = 3$, then $y^i = y^j = y^3$. Therefore, when we multiply $a(y)$ with y^i, we get:

$$\begin{aligned} a(y) * y^i &= a(y) * y^3 \\ &= (a_3y^3 + a_2y^2 + a_1y + a_0) * y^3 \\ &= a_3y^6 + a_2y^5 + a_1y^4 + a_0y^3. \end{aligned}$$

Using the fact that in this arithmetic $y^4 = 1$, we can reduce this equation to:

$$\begin{aligned} a(y) * y &= a_3y^2 + a_2y + a_1 + a_0y^3 \\ &= a_0y^3 + a_3y^2 + a_2y + a_1. \end{aligned}$$

Therefore, $a(y) * y^3 = a(y) = (a_0, a_3, a_2, a_1)$, which is a left circular shift of a by 3 bytes. Thus, in this case the statement is proven true.

From this, we can see that the statement holds for all cases. Therefore, the statement is true.

(b) i. In the Rijndahl field GF(2^8), the bytes (01), (02), and (03) are, respectively:

$$\begin{aligned} c_1(x) &= 1 \\ c_2(x) &= x \\ c_3(x) &= x + 1. \end{aligned}$$

ii. From the previous part, we know that the Rijndahl representation of (02) is $c_2(x) = x$. The representation of b in the Rijndahl field GF(2^8), $b(x)$, is:

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0.$$

Therefore, the value of $d = (02)b$ in the Rijndahl field can be computed as:

$$\begin{aligned} d &= (02)b \\ d(x) &= c_2(x)b(x) \\ &= (x)(b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0) \\ &= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x. \end{aligned}$$

We are told that in this field, arithmetic is done modulo $m(x)$, where $m(x) = x^8 + x^4 + x^3 + x + 1$. Since all math in this field is done modulo $m(x)$, it stands to reason that $m(x) = 0$ in this field, since $m(x) \mod m(x) = 0$. Since $m(x) = 0 = x^8 + x^4 + x^3 + x + 1$, we find that $x^8 = x^4 + x^3 + x + 1$ in this field. We can substitute this into the expression determined for $d = (02)b$ to obtain:

$$\begin{aligned} d(x) &= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \\ &= b_7(x^4 + x^3 + x + 1) + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \\ d(x) &= b_6x^7 + b_5x^6 + b_4x^5 + (b_3 + b_7)x^4 + (b_2 + b_7)x^3 + b_1x^2 + (b_0 + b_7)x + b7. \end{aligned}$$

Thus, we have determine the expression for $d(x)$. Given that d is a byte in the form $d = (d_7d_6d_5 \ldots d_1d0)$, we can write the symbolic expression for each bit d_i of d in terms of the bits of b:

$$\begin{aligned} d_7 &= b_6 \\ d_6 &= b_5 \\ d_5 &= b_4 \\ d_4 &= b_3 + b_7 \\ d_3 &= b_2 + b_7 \\ d_2 &= b_1 \\ d_1 &= b_0 + b_7 \\ d_0 &= b_7. \end{aligned}$$

iii. From the part (i), we know that the Rijndahl representation of (03) is $c_3(x) = x + 1$. The representation of b in the Rijndahl field GF(2^8), $b(x)$, is:

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0.$$

Therefore, the value of $e = (03)b$ in the Rijndahl field can be computed as:

$$\begin{aligned} e &= (03)b \\ e(x) &= c_3(x)b(x) \\ &= (x + 1)(b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0) \\ &= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x + \\ &\quad b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0 \\ &= b_7x^8 + (b_6 + b_7)x^7 + (b_5 + b_6)x^6 + (b_4 + b_5)x^5 + (b_3 + b_4)x^4 + (b_2 + b_3)x^3 + \\ &\quad (b_1 + b_2)x^2 + (b_0 + b_1)x^1 + b_0. \end{aligned}$$

We are told that in this field, arithmetic is done modulo $m(x)$, where $m(x) = x^8 + x^4 + x^3 + x + 1$. We can use the fact that the modulus for a given modular arithmetic is always zero for the corresponding modular arithmetic to determine that $m(x) = 0$, since $m(x)$ is the modulus that corresponds to the Rijndahl field

GF(2^8). Since $m(x) = 0 = x^8 + x^4 + x^3 + x + 1$, we find that $x^8 = x^4 + x^3 + x + 1$ in this field. We can substitute this into the expression determined for $e = (03)b$ to obtain:

$$\begin{aligned}
e(x) &= b_7x^8 + (b_6 + b_7)x^7 + (b_5 + b_6)x^6 + (b_4 + b_5)x^5 + (b_3 + b_4)x^4 + (b_2 + b_3)x^3 + \\
&\quad (b_1 + b_2)x^2 + (b_0 + b_1)x^1 + b_0 \\
&= b_7(x^4 + x^3 + x + 1) + (b_6 + b_7)x^7 + (b_5 + b_6)x^6 + (b_4 + b_5)x^5 + (b_3 + b_4)x^4 + \\
&\quad (b_2 + b_3)x^3 + (b_1 + b_2)x^2 + (b_0 + b_1)x^1 + b_0 \\
e(x) &= (b_6 + b_7)x^7 + (b_5 + b_6)x^6 + (b_4 + b_5)x^5 + (b_3 + b_4 + b_7)x^4 + (b_2 + b_3 + b_7)x^3 + \\
&\quad (b_1 + b_2)x^2 + (b_0 + b_1 + b_7)x^1 + (b_0 + b_7).
\end{aligned}$$

Thus, we have determine the expression for $e(x)$. Given that e is a byte in the form $e = (e_7e_6e_5 \dots e_1e0)$, we can write the symbolic expression for each bit e_i of e in terms of the bits of b:

$$\begin{aligned}
e_7 &= b_6 + b_7 \\
e_6 &= b_5 + b_6 \\
e_5 &= b_4 + b_5 \\
e_4 &= b_3 + b_4 + b_7 \\
e_3 &= b_2 + b_3 + b_7 \\
e_2 &= b_1 + b_2 \\
e_1 &= b_0 + b_1 + b_7 \\
e_0 &= b_0 + b_7.
\end{aligned}$$

(c) i. From part (b), we know that $c(y) = (03)y^3 + (01)y^2 + (01)y + (02)$. Therefore, given that $s(y) = s_3y^3 + s_2y^2 + s_1y + s_0$, we can compute $t(y) = s(y)c(y)$ as:

$$\begin{aligned}
t(y) &= s(y)c(y) \\
&= (s_3y^3 + s_2y^2 + s_1y + s_0)((03)y^3 + (01)y^2 + (01)y + (02)) \\
&= (03)s_3y^6 + (03)s_2y^5 + (03)s_1y^4 + (02)s_0y^3 + \\
&\quad (01)s_3y^5 + (01)s_2y^4 + (01)s_1y^3 + (01)s_0y^2 + \\
&\quad (01)s_3y^4 + (01)s_2y^3 + (01)s_1y^2 + (01)s_0y + \\
&\quad (02)s_3y^3 + (02)s_2y^2 + (02)s_1y + (02)s_0.
\end{aligned}$$

From the previous parts, we know that in this arithmetic, $M(y) = y^4 + 1 = 0$, meaning that $y^4 = 1$. Using this fact, we can reduce the above expression for $t(y)$:

$$
\begin{aligned}
t(y) &= s(y)c(y) \\
&= (s_3y^3 + s_2y^2 + s_1y + s_0)((03)y^3 + (01)y^2 + (01)y + (02)) \\
&= (03)s_3y^6 + (03)s_2y^5 + (03)s_1y^4 + (02)s_0y^3 + \\
&\quad (01)s_3y^5 + (01)s_2y^4 + (01)s_1y^3 + (01)s_0y^2 + \\
&\quad (01)s_3y^4 + (01)s_2y^3 + (01)s_1y^2 + (01)s_0y + \\
&\quad (02)s_3y^3 + (02)s_2y^2 + (02)s_1y + (02)s_0 \\
&= (03)s_3y^2 + (03)s_2y + (03)s_1 + (02)s_0y^3 + \\
&\quad (01)s_3y + (01)s_2 + (01)s_1y^3 + (01)s_0y^2 + \\
&\quad (01)s_3 + (01)s_2y^3 + (01)s_1y^2 + (01)s_0y + \\
&\quad (02)s_3y^3 + (02)s_2y^2 + (02)s_1y + (02)s_0 \\
&= [(03)s_0 + (01)s_1 + (01)s_2 + (02)s_3]y^3 + \\
&\quad [(01)s_0 + (01)s_1 + (02)s_2 + (03)s_3]y^2 + \\
&\quad [(01)s_0 + (02)s_1 + (03)s_2 + (01)s_3]y + \\
&\quad [(02)s_0 + (03)s_1 + (01)s_2 + (01)s_3]
\end{aligned}
$$

The polynomial $t(y) = t_3y^3 + t_2y^2 + t_1y + t_0$ can be written has a vector (t_0, t_1, t_2, t_3). The symbolic expression for each byte in this vector, in terms of the bytes of $s(y)$, can be written as:

$$
\begin{aligned}
t_0 &= (02)s_0 + (03)s_1 + (01)s_2 + (01)s_3 \\
t_1 &= (01)s_0 + (02)s_1 + (03)s_2 + (01)s_3 \\
t_2 &= (01)s_0 + (01)s_1 + (02)s_2 + (03)s_3 \\
t_3 &= (03)s_0 + (01)s_1 + (01)s_2 + (02)s_3.
\end{aligned}
$$

ii. Using the symbolic expressions of the vector (t_0, t_1, t_2, t_3) from the previous part, we can derive the matrix C such that:

$$
\begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} = C \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}.
$$

The matrix C is:

$$
C = \begin{bmatrix} (02) & (03) & (01) & (01) \\ (01) & (02) & (03) & (01) \\ (01) & (01) & (02) & (03) \\ (03) & (01) & (01) & (02) \end{bmatrix}
$$

.

Problem 2 — Error propagation in block cipher modes (12 marks)

(a) i. In ECB, each block of the ciphertext is decrypted individually ($D_K(C_i) = M_i$). The ciphertext blocks have no bearing on the key either; they're completely independent. Therefore, an error in C_i will only affect the plaintext block M_i.

ii. In CBC, each block of the ciphertext is decrypted using the both the current ciphetext block and the previous ciphertext block ($M_i = D_K(C_i) \oplus C_{i-1}$). Therefore, when C_i is being decrypted, its error is propagated to M_i, since $M_i = D_K(C_i) \oplus C_{i-1}$ and C_i has an error. In addition to this, the error in C_i is also propagated to message block M_{i+1}, since $M_{i+1} = D_K(C_{i+1}) \oplus C_i$. Therefore, an error in ciphertext block C_i propagates to message blocks M_i and M_{i+1} in CBC.

iii. In OFB, each block of the ciphertext is decrypted by XORing the ciphertext block with a pseudorandom key stream ($M_i = C_i \oplus KS_i$). The initial key stream (KS_0) is assigned a random initial value. Every subsequent key steam KS_i is generated through the equation $KS_i = E_K(KS_{i-1})$. The ciphertext blocks themselves are independent from the generation of the key stream values. Therefore, an error in C_i will only affect message block M_i.

iv. In CFB with one register, each block of the ciphertext is decrypted by XORing the ciphertext block with a pseudorandom key stream ($M_i = C_i \oplus KS_i$). The initial key stream KS_0 is generated by encrypting some random initial value IV ($KS_0 = E_K(IV)$). Every subsequent key stream KS_i is generated using the ciphertext block from the previous decryption ($KS_i = E_K(C_{i-1})$). An error in C_i will result in an error in the decryption of M_i, since it depends on the value of C_i ($M_i = C_i \oplus KS_i$). Furthermore, an error in C_i will result in an error in the generation of KS_{i+1}, since $KS_{i+1} = E_K(C_i)$. Since KS_{i+1} has an error, this means that the decryption of message block M_{i+1} will also contain an error, since $M_{i+1} = C_{i+1} \oplus KS_{i+1}$. Therefore, an error in ciphertext block C_i will result in errors in the decryptions of both message block M_i and message block M_{i+1}.

v. In CTR, each block of the ciphertext is decrypted by XORing the ciphertext block with a pseudorandom key stream ($M_i = C_i \oplus KS_i$). Each value of KS_i is generate by encrypting a counter value CTR_i of the same size as the plaintext block size ($KS_i = E_K(CTR_i)$). Each subsequent counter value CTR_{i+1} is generated using the previous counter value CTR_i. This means that the generation of key streams is independent from the ciphertext blocks. Therefore, an error in C_i will only result in an error in message block M_i, since $M_i = C_i \oplus KS_i$.

(b) In CBC, each block of the plaintext is encrypted using both the current message block M_i and the last generated ciphertext block C_{i-1}, or the initial value IV (C_0). If there is an error in message block M_i, this will propagate to ciphertext C_i, since $C_i = E_K(M_i \oplus C_{i-1})$. Furthermore, since C_i has errors, this results in C_{i+1} also having errors, since $C_{i+1} = E_K(M_i \oplus C_i)$. The error in C_{i+1} also propagates to C_{i+2}, for the same reason. Therefore, if there is an error in M_i during encryption, this error will propagate into $C_i, C_{i+1}, \ldots, C_n$, where C_n is the final ciphertext block generated during encryption.

Since there's an error in C_i, that means that during decryption, there will be an error in message block M_i, since during decryption ($M_i = D_K(C_i) \oplus C_{i-1}$). However, for the remaining message blocks $M_{i+1}, M_{i+2}, \ldots, M_n$, there is no error during decryption of these message blocks. The decryption of M_{i+1}, involves the equation $M_{i+1} = D_K(C_{i+1}) \oplus C_i$. However, from the encryption process, we know that $C_{i+1} = E_K(M_{i+1} \oplus C_i)$. If we substitute this into the decryption equation, we see that:

$$\begin{aligned} M_{i+1} &= D_K(C_{i+1}) \oplus C_i \\ &= D_K(E_K(M_{i+1} \oplus C_i)) \oplus C_i \\ &= M_{i+1} \oplus C_i \oplus C_i \\ &= M_{i+1}. \end{aligned}$$

From this, we can see that in spite of the ciphertext block having an error introduced during the encryption phase, it doesn't matter during the decryption, since the error that's been introduced is removed when the ciphertext block is XOR'd against itself (since any value XOR'd with itself equals 0). The only reason that M_i contains an error after decryption is because the original message block M_i during encryption also contained an error. Since the error for this message block originated in the plaintext, it remains after decryption. However, for every subsequent block in the encryption process, the error was only introduced during the generation of the ciphertext block, so the error doesn't persist upon decryption.

Problem 3 — Binary exponentiation (13 marks)

(a) For this part, we're asked to perform the exponentiation algorithm, which is applied to evaluate expressions of the form $a^n \pmod m$, to evaluate $17^{11} \bmod (77)$.

First, we must compute the binary representation of 11. We're trying to represent 11 in the form $11 = b_0 2^k + b_1 2^{k-1} + \ldots + b_{k-1} 2 + b_k$, where $b_0 = 1$ and $b_i \in 0, 1$ for $1 \leq i \leq k$, and $k = \lfloor log_2 n \rfloor$. $k = \lfloor log_2(11) \rfloor = 3$. 11 can be represented as $(1)2^3 + (0)2^2 + (1)2 + (1)1$. Therefore, $b_0 = 1, b_1 = 0, b_2 = 1, b_3 = 1$.

Next, we initialize the value r_0 such that $r_0 \equiv a \bmod (m)$. In this case, $r_0 = 17 \pmod{77}$.

Next, for a given r_i, we can compute r_{i+1} using the following rules:

If $b_{i+1} = 0$**, then** $r_{i+1} = r_i^2 \pmod m$

If $b_{i+1} = 1$**, then** $r_{i+1} = r_i^2 a \pmod m$

This process is repeated for $0 \leq i \leq k - 1$. Since $k = 3$ in this case, that means this process is repeated for $0 \leq i \leq 2$. The process is as follows:

For i = 0, $b_{i+1} = b_{0+1} = b1 = 0$. Therefore, since $b_1 = 0$:

$$\begin{aligned} r_{0+1} &= r_1 \\ &= r_0^2 \mod 77 \\ r_1 &= 17^2 \mod 77 \\ &= 289 \mod 77 \\ r_1 &= 58. \end{aligned}$$

For i = 1, $b_{i+1} = b_{1+1} = b2 = 1$. Therefore, since $b_2 = 1$:

$$\begin{aligned} r_{1+1} &= r_2 \\ &= r_1^2 a \mod 77 \\ r_2 &= 58^2 * 17 \mod 77 \\ &= 57188 \mod 77 \\ r_2 &= 54. \end{aligned}$$

For i = 2, $b_{i+1} = b_{2+1} = b3 = 1$. Therefore, since $b_3 = 1$:

$$
\begin{aligned}
r_{2+1} &= r_3 \\
&= r_2^2 a \mod 77 \\
r_3 &= 54^2 * 17 \mod 77 \\
&= 49572 \mod 77 \\
r_3 &= 61.
\end{aligned}
$$

Therefore, at the end of our approach, we find that $r_3 = 61$. Therefore, this means that $17^{11} \mod 77 = 61$.

(b) i. In this question, we are asked to prove that $s_i = \sum_{j=0}^{i} b_j 2^{i-j}$ for $0 \leq i \leq k$, given that $s_0 = 1$ and $s_{i+1} = 2s_i + b_{i+1}$ for $0 \leq i \leq k-1$. We shall prove this using induction.

Base Case i = 0: If $i = 0$, then $s_0 = \sum_{j=0}^{0} b_j 2^{i-j} = b_0 2^{0-0} = b_0 2^0 = b_0$. In the description of the exponentiation algorithm, the value of b_0 is defined as being $b_0 = 1$. Therefore, $s_0 = b_0 = 1$. This matches the value that we defined for s_0 in the question. Thus, the base case has been verified.

Inductive Hypothesis: Suppose that $s_i = \sum_{j=0}^{i} b_j 2^{i-j}$ for $0 \leq i \leq k-1$.

Inductive Step: We want to show that this inductive hypothesis also holds for $i+1$. From the question, we are provided with the definition that $s_{i+1} = 2s_i + b_{i+1}$ for $0 \leq i \leq k$. From the inductive hypothesis we know that $s_i = \sum_{j=0}^{i} b_j 2^{i-j}$. We can substitute this into the previous equation to get: $s_{i+1} = 2(\sum_{j=0}^{i} b_j 2^{i-j}) + b_{i+1}$. The 2 multiplying the summation can be moved into the equation to get the following: $s_{i+1} = \sum_{j=0}^{i} b_j 2^{i-j} 2 + b_{i+1}$. Using power rules, we can re-write this into $s_{i+1} = \sum_{j=0}^{i} b_j 2^{i+1-j} + b_{i+1}$. The term b_{i+1} can be re-written as $b_{i+1} 2^0 = b_{i+1} 2^{i+1-(i+1)}$. This given us:

$$
s_{i+1} = \sum_{j=0}^{i} b_j 2^{i+1-j} + b_{i+1} 2^{i+1-(i+1)}
$$

The second term is of the form $b_j 2^{i+1-j}$ where $j = i+1$. This means that it can be added to the summation by increasing the upper limit on the summation from i to $i+1$. Finally, we see that:

$$
s_{i+1} = \sum_{j=0}^{i+1} b_j 2^{i+1-j}.
$$

Thus, we have proven that the inductive hypothesis applies to $i+1$.

ii. In this question, we are asked to prove that $r_0 \equiv a \mod m$ for $0 \leq i \leq k$, given the definitions provided in steps 2 and 3 of the exponentiation algorithm. We will prove this using induction.

Base Case i = 0: If $i = 0$, then we get $r_0 \equiv a^{s_0} \mod m$. From the previous part, we know that $s_0 = 1$. Therefore, this equation becomes: $r_0 \equiv a \mod m$. This is identical to the definition provided in step 2 of the exponentiation algorithm. Therefore, the base case is proven to be verified.

Inductive Hypothesis: Suppose that $r_i \equiv a^{s_i}$ for $0 \leq i \leq k-1$.

Inductive Step: We want to prove that $r_{i+1} \equiv a^{s_{i+1}} \mod m$. We can compute s_{i+1} using the definition provided in the previous part: $s_{i+1} = 2s_i + b_{i+1}$. Since the value of b_{i+1} is unknown, there are two cases that must be considered.

Case 1, $b_{i+1} = 0$: If $b_{i+1} = 0$, then $s_{i+1} = 2s_i + 0 = 2s_i$. Therefore, $r_{i+1} \equiv a^{2s_i} \mod m \equiv (a^{s_i})^2 \mod m$. From the inductive hypothesis, we know that $r_i \equiv a^{s_i} \mod m$. Therefore, when $b_{i+1} = 0$, then $r_{i+1} \equiv r_i^2 \mod m$. This is consistent with what is show in step 3 of the exponentiation algorithm when $b_{i+1} = 0$. Thus, we have proven that $r_{i+1} \equiv a^{s_{i+1}}$ in this case.

Case 2, $b_{i+1} = 1$: If $b_{i+1} = 1$, then $s_{i+1} = 2s_i$. Therefore, $r_{i+1} \equiv a^{2s_i+1} \mod m \equiv a^{2s_i}a \mod m \equiv (a^{s_i})^2 a \mod m$. From the inductive hypothesis, we know that $r_i \equiv a^{s_i} \mod m$. Therefore, when $b_{i+1} = 1$, then $r_{i+1} \equiv r_i^2 a \mod m$. This is consistent with what is show in step 3 of the exponentiation algorithm when $b_{i+1} = 1$. Thus, we have proven that $r_{i+1} \equiv a^{s_{i+1}}$ in this case.

Since we have proven that $r_{i+1} \equiv a^{s_{i+1}}$, in both cases, the statement holds in all cases. Therefore, we have proven that the inductive hypothesis applies to $i + 1$.

iii. We are asked to prove that $a^n \equiv r_k$. In part (i), we proved that $s_i = \sum_{j=0}^{i} b_j 2^{i-j}$ for $0 \leq i \leq k$. If we set $i = k$, we find that $s_k = \sum_{j=0}^{k} b_j 2^{i-j}$. When we expand this, we find that:

$$\begin{aligned} s_k &= b_0 2^{k-0} + b_1 2^{k-1} + \ldots + b_{k-1} 2^{k-(k-1)} + b_k 2^{k-k} \\ &= b_0 2^k + b_1 2^{k-1} + \ldots + b_{k-1} 2 + b_k. \end{aligned}$$

This expression for s_k perfectly matches step one of the exponentiation algorithm, where the exponent n in the expression $a^n \mod m$ is written in the form: $n = b_0 2^k + b_1 2^{k-1} + \ldots + b_{k-1} 2 + b_k$. From this, we can conclude that $s_k = n$.

Next, in part (ii), we proved that $r_i \equiv a^{s_i}$ for $0 \leq i \leq k$. If we set $i = k$, we can see that $r_k = a^{s_k} \mod m$. Earlier, we found that $s_k = n$. If we substitute this equality in, we see that $r_k \equiv a^n \mod m$. This can be re-written as $a^n \equiv r_k \mod m$. Thus, we have proven that $a^n \equiv r_k \mod m$, thereby verifying that the exponentiation algorithm does compute $a^n \mod m$ like it claims.

Problem 4 — A modified man-in-the-middle attack on Diffie-Hellman (10 marks)

(a) Mallory sends Alice $(g^b)^q \mod m$. She then computes Key_{Alice} as $((g^b)^q)^a \mod m = g^{abq} \mod m$. Therefore, $Key_{Alice} = g^{abq} \mod m$.

Mallory sends Bob $(g^a)^q \mod m$. He then computes Key_{Bob} as $((g^a)^q)^b \mod m = g^{abq} \mod m$. Therefore, $Key_{Bob} = g^{abq} \mod m$.

From this, we can see that $Key_{Alice} = Key_{Bob}$. Therefore, both Alice and Bob end up calculating the same shared key.

(b) In this question, we are asked to demonstrate that in the case of Mallory's attack, there are only m possible values for K.

In the question, we are told that p is prime, and that g is a primitive root of p. Thus, by Fermat's Little Theorem, we know that $g^{p-1} \equiv 1 \mod p$. Furthermore, we are told that $p = mq + 1$, where q is also a prime number, and m is noted as being very small. We can rearrange this equation to obtain $mq = p - 1$.
From the previous part, we know that the shared key between Alice and Bob, K, can be represented by $g^{abq} \equiv K \mod p$. Mallory herself knows both $g^{aq} \mod p$ and $g^{bq} \mod p$, and is trying to find $g^{abq} \mod p$. Suppose she chooses to try finding it using the known value $g^{aq} \mod p$. Let the integer k, $k \geq 0$, represent the candidate values for b. Using this, we get: $g^{kaq} \mod p$, $k \geq 0$.

When $k = 0$, this expression becomes $g^{kaq} \equiv g^0 \equiv 1 \mod p$.
When $k = 1$, this expression becomes $g^{kaq} \equiv g^{aq} \mod p$.
When $k = 2$, this expression becomes $g^{kaq} \equiv g^{2aq} \mod p$.
$\vdots$
Suppose we follows trends all the up to $k = m$. Then we get $g^{maq} \equiv K \mod p$. This can be re-arranged to get $(g^{mq})^a \equiv K \mod p$. However, we know that $mq = p - 1$ from above, meaning this can be re-written into $(g^{p-1})^a \equiv K \mod p$. Since g is a primitive root of p, we know from Fermat's Little Theorem that $g^{p-1} \equiv 1 \mod p$. That means that $(g^{p-1})^a \mod p$ can be replaced with $1^a \mod p$, which is equivalent to $1 \mod p$. Therefore, when $k = m$, the expression can be reduced to $1 \equiv K \mod p$. This is identical to the expression we got for $k = 0$. Furthermore, for each subsequent value of k ($k = m + 1$, $k = m + 2$, $k = m + 3$ ...), there will always be a multiple of m within the value, which can be extracted and then reduced to 1 via Fermat's Little Theorem. The portion of k that remains after this process will be some value r such that we get $g^r \mod p$, $0 \leq r \leq m - 1$, which all have computed values already. Therefore, Mallory only needs to test m values to find K. Since m is known to be small, that means there aren't many numbers to test.

(c) In the man-in-the-middle attack discussed in lecture, there are two shared keys: $g^{ea} \mod m$, which is shared between Alice and Mallory, and then there is $g^{eb} \mod m$, which is shared between Mallory and Bob. In this scenario, Mallory has to intercept every message sent by Alice, decrypt it using the key she shares with Alice, and then re-encrypt the message with the key she shares with Bob, and vice versa for the messages that Bob sends. If she fails to do this a single message, then Alice and Bob may notice

that there is an interceptor. The reason is because Alice and Bob don't have a shared key with each other. This means that if Alice gets a message directly to Bob without Mallory intercepting it, Bob won't be able to decrypt it. Therefore, this may indicate to them that something suspicious is happening. This approach requires much more effort from Mallory, since she can't afford to miss a single message, lest she be found out.

In comparison, the attack explored in this question is far more passive. Mallory doesn't need to worry about intercepting every message, because Alice and Bob *do* share a key in this case. Therefore, even if Mallory misses intercepting a message, the recipient can still decrypt it successfully. This greatly reduces the chances of her being caught.

Problem 5 — A simplified password-based key agreement protocol (8 marks)

(a) By the end of this procedure, $K_{Client} \equiv B^{a+p} \mod N$. However, we know from step 1 of the protocol that $B \equiv g^b \mod N$. Therefore, we can re-write K_{Client} as $(g^b)^{a+p} \equiv \mod N$. This can be expanded out to obtain: $K_{Client} \equiv g^{ab+bp} \mod N$.

By the end of this procedure, $K_{Server} \equiv (Av)^b \mod N$. However, we know from the question's description of the protocol that $v \equiv g^p \mod N$. We also know from step one that $A \equiv g^a \mod N$. Therefore, we can re-write K_{Server} as $K_{Server} \equiv (g^a g^p)^b \mod N$. This can be expanded out to obtain: $K_{Server} \equiv (g^{a+p})^b \mod N \equiv g^{ab+bp} \mod N$. Therefore, $K_{Server} = g^{ab+bp} \mod N$.

From this, we can see that $K_{Client} \equiv g^{ab+bp} \mod N$, and that $K_{Server} \equiv g^{ab+bp} \mod N$. Therefore, $K_{Client} = K_{Server}$.

(b) Suppose that the value v is known by a malicious user. Let us then choose $a = (N-2)p$ for our value. If we do this, then $A \equiv g^{(N-2)p} \mod N$. Since g is a primitive root of N, that means that $g^{N-1} \equiv 1 \mod N$ according to Fermat's Little Theorem. We can write $a = (N-1-1)p = (N-1)p - p$. Thus, we can see that $A \equiv g^{(N-1)p-p} \mod N$. Using exponent rules, this can be written as $A \equiv (g^{N-1})^p * g^{-p} \mod N$. Since $g^{N-1} \equiv 1 \mod N$, This becomes $A \equiv (1)^p * g^{-p} \mod N \equiv g^{-p} \mod N$. Thus, $A \equiv g^{-p} \mod N$. The server computes K_{Server} as $(Av)^b \mod N$. Since the value of A being sent over is $g^{-p} \mod N$. This computation becomes $((g^{-p})v)^b \mod N$. Since $v \equiv g^p \mod N$, this can be re-written as: $(g^{-p} * g^p) \mod N$. Using exponent rules, this becomes $g^{-p+p} \mod N = g^0 \mod N = 1 \mod N$. Therefore, $K_{Server} \equiv 1 \mod N$.

The client calculates its key, K_{Client}, using $K_{Client} \equiv (B)^{a+p} \mod N$. Since we've defined $a = (N-1)p - p$, we can re-write this as: $K_{Client} \equiv (B)^{(N-1)p-p+p} \mod N \equiv (B)^{(N-1)p} \mod N$. Since $K_{Client} \equiv (B)^{(N-1)p} \mod N$, equation becomes $K_{Client} \equiv (g^b)^{(N-1)p} \mod N$. This can be re-arranged to get: $K_{Client} \equiv (g^{N-1})^{bp} \mod N$. Since g is a primitive root of N, that means that $g^{N-1} \equiv 1 \mod N$, by Fermat's Little Theorem. Therefore, this equation can be turned into $K_{Client} \equiv (1)^{bp} \mod N \equiv 1 \mod N$. Thus, $K_{Client} \equiv 1 \mod N$. This matches the value of K_{Server}. Thus, we have shown that with knowledge of v, we can always select a value of A such that we can successfully masquerade as the client.

(c) For this question, we are asked to prove that a user who can solve any instance of the key recovery problem for the protocol above can also solve any instance of the Diffie-Hellman problem as well. Suppose a user can solve the key recovery problem above. That means that they can find any shared key of the form $K = g^{ab+bp}$.

The Diffie-Hellman protocol generates a shared key of the form $K = g^{ab}$. If we compare a Diffie-Hellman key to the one generated by the protocol laid out in this question, we can see that the Diffie-Hellman key is just an instance of form g^{ab+bp} where $p = 0$ ($g^{ab+b(0)} = g^{ab+0} = g^{ab}$). Therefore, since the user can find any shared key of the form g^{ab+bp}, and the Diffie-Hellman problem generates a shared key of the form g^{ab+bp} where $p = 0$, it stands to reason that the user can also solve the Diffie-Hellman problem.