CPSC 418 / MATH 318 — Introduction to Cryptography ASSIGNMENT 2

Name: Karmvir Singh Dhaliwal
Student ID: 30025474

Problem 1 — Arithmetic in the AES MixColumns operation

a. (i) Suppose we have a 4-byte vector, viewed as a polynomial *M'* where the coefficients are bytes. We have:

$$M' = a_3y^3 + a_2y^2 + a_1y + a_0 \tag{1}$$

Where a_3, a_2, a_1, a_0 are all bytes. If we multiply by y, we get:

$$\begin{aligned}(M')(y) &= (a_3y^3 + a_2y^2 + a_1y + a_0)(y) \\ &= a_3y^4 + a_2y^3 + a_1y^2 + a_0y\end{aligned} \tag{2}$$

We can then reduce this polynomial by mod *M(y)* = $y^4 + 1$, which gives us:

$$\begin{aligned}a_3y^4 + a_2y^3 + a_1y^2 + a_0y \quad \bmod y^4 + 1& \\ &= a_3 + a_2y^3 + a_1y^2 + a_0y \\ &= a_2y^3 + a_1y^2 + a_0y + a_3\end{aligned} \tag{3}$$

Which we can see is the original polynomial but with all of the coefficients, which are bytes, moved one position to the left. In other words, all of the bytes have been circular shifted one bit to the left, as required. As we have the desired result, we have formally proven that multiplication of any 4-byte vector by y is a circular left shift of the vector by one byte. □

(ii) From part *a(i)*, we know that a multiplication of any 4-byte vector by y is a circular left shift of the vector by one byte. We want to prove that multiplication of any 4-byte vector by y^i $(0 \leq i \leq 3)$ is a circular left shift of the vector by i bytes. Now, observe that y^i can be written as:

$$y^i = y \times ... \times y \quad \text{- i times} \tag{4}$$

In other words, we can split y^i into repeated multiplication of y, i times. Since multiplication is associative, we can rewrite the multiplication of our polynomial M by y^i as:

$$(M)(y^i) = (((M \times y) \times y)... \times y) \quad \text{- with } i \text{ y's.} \tag{5}$$

In other words, we multiply our original polynomial M by y to get a new polynomial M' which we multiply by y to get M", and this continues until we have multiplied by i y's. Each multiplication of y will give us a circular shift left by 1 byte, again as proven in part *a(i)*. In total, we will have a circular shift left by i bytes, since each multiplication by y gives us a circular shift left by 1 byte, and we have i total multiplications by y, as required. Thereby proving that multiplication of any 4-byte vector by $y^i(0 \leq i \leq 3)$ is a circular left shift of the vector by i bytes. □

b. (i) The hexadecimal byte 01 in binary is 00000001. The polynomial representation of this is x^0, or just 1.
The hexadecimal byte 02 in binary is 00000010. The polynomial representation of this is x^1 or just x.
The hexadecimal byte 03 in binary is 00000011. The polynomial representation of this is $x^1 + x^0$ or $x + 1$.

(ii) Let b be a byte $(b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0)$. The polynomial representing b is:

$$b_{(x)} = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \tag{6}$$

From above, we know that hexadecimal 02 in polynomial form in this field is simply x. This means our multiplication is:

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

We now have two cases, depending on the value of b_7. Also, if b is a bit, let -b represent the opposite of the bit. So, if b = 1 -b = 0, and vice versa. Now:

i. Case 1: $b_7 = 1$: If b_7 is one, then we must reduce the polynomial by:

$$M_{(x)} = x^8 + x^4 + x^3 + x + 1. \tag{8}$$

This gives us the polynomial:

$$b_6x^7 + b_5x^6 + b_4x^5 + -b_3x^4 + -b_2x^3 + b_1x^2 + -b_0x + 1 \tag{9}$$

So, the equations for the bits d_7 - d_0 are:

$$\begin{aligned} d_7 &= b_6 \\ d_6 &= b_5 \\ d_5 &= b_4 \\ d_4 &= -b_3 \\ d_3 &= -b_2 \\ d_2 &= b_1 \\ d_1 &= -b_0 \\ d_0 &= 1 \end{aligned} \tag{10}$$

The reason d_4, d_3 and d_1 are the opposites of b_3, b_2 and b_0 respectively, is because of the modular reduction. If for example, b_3 is 1, then b_3x^4 would cancel out during the modular reduction; if it is 0 it wouldn't cancel. The same can be said about b_2 and b_0.

ii. Case 2: $b_7 = 0$: If $b_7 = 0$, then we don't need to reduce the polynomial. This gives us the polynomial:

$$b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \tag{11}$$

So, the equations for the bits d_7 - d_0 are:

$$\begin{aligned} d_7 &= b_6 \\ d_6 &= b_5 \\ d_5 &= b_4 \\ d_4 &= b_3 \\ d_3 &= b_2 \\ d_2 &= b_1 \\ d_1 &= b_0 \\ d_0 &= 0 \end{aligned} \tag{12}$$

(iii) Let b be a byte $(b_7b_6b_5b_4b_3b_2b_1b_0)$. The polynomial representing b is:

$$b_{(x)} = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \tag{13}$$

From above, we know that hexadecimal 03 in polynomial form in this field is simply $(x+1)$. This means our multiplication is:

$$(03)b = (x+1)(b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0) \tag{14}$$

$$\begin{aligned} &= b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x \\ &\quad + b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \end{aligned}$$

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

Again, we have 2 cases:

i. Case 1: $b_7 = 1$: If b_7 is one, then we must reduce the polynomial by:

$$M_{(x)} = x^8 + x^4 + x^3 + x + 1. \tag{15}$$

This gives us the polynomial:

$$\begin{aligned} &(b_6+b_7)x^7 + (b_5+b_6)x^6 + (b_4+b_5)x^5 + -(b_3+b_4)x^4 + -(b_2+b_3)x^3 \\ &\qquad + (b_1+b_2)x^2 + -(b_0+b_1)x + (b_0+1) \end{aligned}$$

So, the equations for the bits d_7 - d_0 are:

$$\begin{aligned} d_7 &= (b_6+b_7) \\ d_6 &= (b_5+b_6) \\ d_5 &= (b_4+b_5) \\ d_4 &= -(b_3+b_4) \\ d_3 &= -(b_2+b_3) \\ d_2 &= (b_1+b_2) \\ d_1 &= -(b_0+b_1) \\ d_0 &= (b_0+1) \end{aligned} \tag{16}$$

Again, we must use the opposite of $(b_3 + b_4), (b_2 + b_3)$ and $(b_0 + b_1)$ because of the modular reduction. If these additions equal 1, they will be cancelled out using the modular reduction; if they equal 0 they will not be cancelled.

ii. Case 2: $b_7 = 0$: If $b_7 = 0$:, we won't need to do the modular reduction. This gives us the polynomial:

$$\begin{aligned}(b_6 + b_7)x^7 + (b_5 + b_6)x^6 + (b_4 + b_5)x^5 + (b_3 + b_4)x^4 \\ + (b_2 + b_3)x^3 + (b_1 + b_2)x^2 + (b_0 + b_1)x + b_0\end{aligned}$$

So, the equations for the bits d_7 - d_0 are:

$$\begin{aligned} d_7 &= (b_6 + b_7) \\ d_6 &= (b_5 + b_6) \\ d_5 &= (b_4 + b_5) \\ d_4 &= (b_3 + b_4) \\ d_3 &= (b_2 + b_3) \\ d_2 &= (b_1 + b_2) \\ d_1 &= (b_0 + b_1) \\ d_0 &= (b_0) \end{aligned} \tag{17}$$

c. (i) Upon initial multiplication of $C_{(y)}$ and $S_{(y)}$ we get:

$$\begin{aligned} S_{(y)} &= s_3 y^3 + s_2 y^2 + s_1 y + s_0 \\ C_{(y)} &= (03)y^3 + (02)y^2 + (01)y + (02) \\ S_{(y)}C_{(y)} &= (s_3 y^3 + s_2 y^2 + s_1 y + s_0)((03)y^3 + (02)y^2 + (01)y + (02)) \end{aligned} \tag{18}$$

$$\begin{aligned} &= s_3(03)y^6 + s_3(02)y^5 + s_3(01)y^4 + s_3(02)y^3 \\ &+ s_2(03)y^5 + s_2(02)y^4 + s_2(01)y^3 + s_2(02)y^2 \\ &+ s_1(03)y^4 + s_1(02)y^3 + s_1(01)y^2 + s_1(02)y \\ &+ s_0(03)y^3 + s_0(02)y^2 + s_0(01)y + s_0(02) \end{aligned}$$

Now, after reduction modulo $M_{(x)} = y^4 + 1$ we get:

$$\begin{aligned} &= s_3(03)y^2 + s_3(02)y + s_3(01) + s_3(02)y^3 \\ &+ s_2(03)y + s_2(02) + s_2(01)y^3 + s_2(02)y^2 \\ &+ s_1(03) + s_1(02)y^3 + s_1(01)y^2 + s_1(02)y \\ &+ s_0(03)y^3 + s_0(02)y^2 + s_0(01)y + s_0(02) \end{aligned}$$

Now all we need to do is collect like terms:

$$\begin{gathered}
= ((s_3(02)) + (s_2(01)) + (s_1(02)) + (s_0(03)))y^3 \\
+ ((s_3(03)) + (s_2(02)) + (s_1(01)) + (s_0(02)))y^2 \\
+ ((s_3(02)) + (s_2(03)) + (s_1(02)) + (s_0(01)))y \\
+ ((s_3(01)) + (s_2(02)) + (s_1(03)) + (s_0(02)))
\end{gathered}$$

From this we can see:

$$\begin{aligned}
t_0 &= ((s_3(01)) + (s_2(02)) + (s_1(03)) + (s_0(02))) \\
t_1 &= ((s_3(02)) + (s_2(03)) + (s_1(02)) + (s_0(01))) \\
t_2 &= ((s_3(03)) + (s_2(02)) + (s_1(01)) + (s_0(02))) \\
t_3 &= ((s_3(02)) + (s_2(01)) + (s_1(02)) + (s_0(03)))
\end{aligned} \tag{19}$$

(ii) The matrix C can be represented as follows:

$$\begin{bmatrix} 02 & 03 & 02 & 01 \\ 01 & 02 & 03 & 02 \\ 02 & 01 & 02 & 03 \\ 03 & 02 & 01 & 02 \end{bmatrix} \tag{20}$$

We get this by simply taking our equations t_0 - t_3 and formatting them so the matrix multiplication will go through correctly, that is we rewrite them in order from $t_0 - t_3$, and for each equation we reorder it from $s_0 - s_3$.

Problem 2 — Error propagation in block cipher modes

a. (i) As we know from class, ECB mode simply encrypts blocks sequentially. Each cipher block C_i is:

$$C_i = E_k(M_i) \quad \text{- For i = 1, 2,...} \tag{21}$$

This means that each block C_i has no bearing on future blocks, so an error in C_i will only cause an error plaintext block M_i.

(ii) As we know from class, in CBC mode encryption of block C_i is:

$$C_i = E_k(M_i \oplus C_{i-1}) \tag{22}$$

This means that block C_i will be used in the encryption of block C_{i+1}, which will in turn be used in the calculation of C_{i+2}, and so on. Essentially, the error will propagate all the way through to the final cipher text block, as each block requires the previous block in encryption. So, the error will be present in every block from M_i until the last plaintext block.

(iii) As discussed in tutorial, in OFB mode the way we encrypt is:

$$C_i = M_i \oplus KS_i \tag{23}$$

Where KS_i is:

$$\begin{cases} IV & i = 0 \\ E_k(KS_{i-1}) & i > 0 \end{cases} \tag{24}$$

As we can see, each block C_i has no bearing on future blocks, so an error in C_i will only cause an error plaintext block M_i.

(iv) As discussed in tutorial, encryption in CFB with one register can be described as

$$C_i = (M_i \oplus KS_i) \tag{25}$$

Where KS_i is:

$$\begin{cases} IV & i = 0 \\ E_k(C_{i-1}) & i > 0 \end{cases} \tag{26}$$

In other words, the encryption of block C_i is used in the encryption of block C_{i+1}. For this reason, the error will propagate all the way through to the final cipher text block, as each block requires the previous block in encryption. So, the error will be present in every block from M_i until the last plaintext block.

(v) Again, as discussed in tutorial we know that encryption in CTR mode can be shown as:

$$C_i = M_i \oplus E_k(nonce||ctr) \tag{27}$$

Where ctr is a counter that is incremented for each block. As we can see, each block C_i has no bearing on future blocks, so an error in C_i will only cause an error plaintext block M_i.

b. Since the error occurred in the message M_i itself and not in the cipher block C_i, C_i will be the correct encryption of message M_i. As discussed in tutorial, the message M_i itself has no bearing on the decryption of future blocks, it is the cipher block C_i that is used in future blocks. For this reason, if there is an error in message block M_i, the only message block affected after decryption will be M_i, since C_i will correctly decrypt the block C_{i+1}, which in turn will correctly decrypt the block C_{i+2} and so on.

Problem 3 — Binary exponentiation

a. Using the given formula we get:

$$\begin{aligned}
&\text{Step 1: get binary representation of n}\\
n &= 11\\
\text{binary representation} &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)\\
&\text{So:}\\
k &= 3 \text{ and,}\\
b_0 &= 1\\
b_1 &= 0\\
b_2 &= 1\\
b_3 &= 1
\end{aligned} \tag{28}$$

Now for step 2:

$$\begin{aligned}
&\text{Initializing } r_0\text{:}\\
r_0 &\equiv a \mod m\\
&= 17 \mod 77\\
r_0 &= 17
\end{aligned} \tag{29}$$

Next, we perform step 3:

$$\begin{aligned}
&\text{Since } b_1 = 0\text{: } r_1 \equiv r_0^2 \mod 77 \equiv 289 \mod 77 \equiv 58 \mod 77\\
&\text{Since } b_2 = 1\text{: } r_2 \equiv r_1^2 \times a \mod 77 \equiv 58^2 \times 17 \mod 77 \equiv 54 \mod 77\\
&\text{Since } b_3 = 1\text{: } r_3 \equiv r_2^2 \times a \mod 77 \equiv 54^2 \times 17 \mod 77 \equiv 61 \mod 77
\end{aligned} \tag{30}$$

And finally we simply output r_k, which we have calculated to be 61 mod 77. So, $17^{11} \mod 77 \equiv 61 \mod 77$.

b. (i) To prove by induction, we begin by proving the base case, that is when i = 0. Now:

$$\begin{aligned}
i &= 0\\
s_0 &= 1 \quad \text{-as defined}\\
&= 1 \times 1\\
&= b_0 \times 1 \quad \text{-We know } b_0 \text{ represents the most significant bit from part (a), so } b_0 \text{ must be 1.}\\
&= b_0 \times 2^0\\
&= b_0 \times 2^{0-0}\\
&= \sum_{j=0}^{i=0} b_j \times 2^{i-j}
\end{aligned} \tag{31}$$

As required. We have proven that the base case holds. Now, suppose this property holds for $0 \leq i \leq k-1$. That is, $s_i = \sum_{j=0}^{i} b_j \times 2^{i-j}$ We want to prove it holds for $i+1$. Now:

$$\begin{aligned}
&s_{i+1} \\
&= 2s_i + b_{i+1} \quad \text{-as defined} \\
&= 2(\sum_{j=0}^{i} b_j \times 2^{i-j}) + b_{i+1} \quad \text{-by inductive hypothesis} \\
&= 2(\sum_{j=0}^{i} b_j \times 2^{i-j}) + 2^0 b_{i+1} \\
&= \sum_{j=0}^{i} b_j \times 2^{i+1-j} + 2^0 b_{i+1} \\
&= \sum_{j=0}^{i} b_j \times 2^{i+1-j} + 2^{i+1-i-1} b_{i+1} \\
&= \sum_{j=0}^{i+1} b_j \times 2^{i+1-j}
\end{aligned} \tag{32}$$

As required, therefore proving using mathematical induction that $s_i = \sum_{j=0}^{i} b_j \times 2^{i-j}$ for $0 \leq i \leq k$. $\square$

(ii) To prove by induction, we begin by proving the base case, when i = 0. Now:

$$\begin{aligned}
i &= 0 \\
r_0 &\equiv a \mod m \quad \text{-From step 2} \\
&= a^1 \mod m \\
&= a^{s_0} \mod m \quad \text{-As proven above}
\end{aligned} \tag{33}$$

As required. Now, assume this property holds for all i such that $0 \leq i \leq k-1$. That is, $r_i \equiv a^{s_i} \mod m$. We prove for i+1. Now:

$$\begin{aligned}
r_{i+1} &= \begin{cases} r_i^2 \mod m & \text{if } b_{i+1} = 0 \\ r_i^2 a \mod m & \text{if } b_{i+1} = 1 \end{cases} \\
&= \begin{cases} a^{s_i^2} \mod m & \text{if } b_{i+1} = 0 \\ a^{{s_i}^2} a \mod m & \text{if } b_{i+1} = 1 \end{cases} \quad \text{-From inductive hypothesis} \\
&= \begin{cases} a^{2s_i} \mod m & \text{if } b_{i+1} = 0 \\ a^{2s_i} a \mod m & \text{if } b_{i+1} = 1 \end{cases} \\
&= \begin{cases} a^{2s_i + b_{i+1} - b_{i+1}} \mod m & \text{if } b_{i+1} = 0 \\ a^{2s_i + 1} \mod m & \text{if } b_{i+1} = 1 \end{cases} \\
&= \begin{cases} a^{2s_i + b_{i+1} - b_{i+1}} \mod m & \text{if } b_{i+1} = 0 \\ a^{2s_i + b_{i+1}} \mod m & \text{if } b_{i+1} = 1 \end{cases} \\
&= \begin{cases} a^{s_{i+1}} \mod m & \text{if } b_{i+1} = 0 \\ a^{s_{i+1}} \mod m & \text{if } b_{i+1} = 1 \end{cases} \\
&= a^{s_{i+1}} \mod m
\end{aligned} \tag{34}$$

As required, thereby proving using mathematical induction that $r_i \equiv a^{s_i} \mod m$ for all i such that $0 \leq i \leq k$. $\square$

(iii) From above, we have that $r_i \equiv a^{s_i} \mod m$. This means that $r_k \equiv a^{s_k} \mod m$. Now:

$$\begin{aligned} s_k &= \sum_{j=0}^{i} b_j 2^{i-j} \\ &= b_0 2^k + b_1 2^{k-1} + ... + b_{k-1}2 + b_k \quad \text{-By simply expanding out the summation} \\ &= n \quad \text{From the question} \end{aligned} \tag{35}$$

So, since $s_k = n$ we have $r_k \equiv a^n \mod m$. From the symmetric property of congruence, we get $a^n \equiv r_k \mod m$ as required. $\square$

Problem 4 — A modified man-in-the-middle attack on Diffie-Hellman

a. We know from the Diffie-Hellman protocol that upon receiving g^a from Alice, Bob will compute $(g^a)^b$. Similarly, upon receiving g^b from Bob, Alice will compute $(g^b)^a$. In this case however, since Alice actually receives $(g^b)^q$, she will compute $(((g^b)^q)^a)$. Similarly again, since Bob actually receives $(g^a)^q$ he will compute $(((g^a)^q)^b)$. Using the power rules we can see that:

$$\begin{aligned} &(((g^b)^q)^a) \\ &= ((g^b)^{qa}) \quad \text{- By power rules} \\ &= g^{bqa} \quad \text{- By power rules} \end{aligned} \tag{36}$$

and:

$$\begin{aligned} &(((g^a)^q)^b) \\ &= ((g^a)^{qb}) \quad \text{- By power rules} \\ &= g^{aqb} \quad \text{- By power rules} \end{aligned} \tag{37}$$

Which, by the associative property of multiplication are equivalent. Thus proving that Alice and Bob compute the same key. □

b. From tutorial, we know that $g^{abq} \equiv g^r \mod p$. From the quotient remainder theorem, we know that $0 \leq r \leq p-2$, but we need another restriction on r. Now, observe:

$$\begin{aligned} abq &= s(p-1)+r \quad \text{-From quotient remainder theorem} \\ abq &= s(mq)+r \quad \text{-Because (p-1) = mq} \\ r &= abq-((s)mq) \\ r &= q(ab-sm) \end{aligned} \tag{38}$$

What this tells us is that r is a multiple of q. Now, we know that $mq \mod p$ will wrap around, so we have that r can be q, 2q, 3q, ... , mq. We have exactly m values that r can be, and they are all easy to calculate, as they are just the multiples of q.

c. The biggest benefit to this attack compared to the one discussed in class is that Alice and Bob compute the same key. In the attack discussed in class, both Alice and Bob have different keys, as Alice uses $((g^e)^a)$ as a key and Bob uses $((g^e)^b)$ as a key. This means that Mallory has to be active in the middle at all times, intercepting and decrypting messages that are sent with one key and re-encrypting with the second key and forwarding messages so that Alice and Bob don't catch on to anything. If at any point Mallory were to stop doing this, the message Alice sent to Bob, or vice versa, would not decrypt correctly, as the key Bob has and the key Alice has are different. This would likely alert Alice and Bob that something was wrong, and the attack could be uncovered. In this attack though, the key is the same. Mallory does not need to constantly intercept, decrypt, encrypt and forward messages, she just needs to decrypt the messages so she can look at them. If Mallory steps aside so the messages are just being sent directly from Alice to Bob and vice versa, everything would still work perfectly fine as both Alice and Bob would have the same key. This makes this attack a lot harder to detect, since everything works just as it should, which makes this attack much more advantageous compared to the one discussed in class.

Problem 5 — A simplified password-based key agreement protocol

a. The client calculates:

$$\begin{aligned} &B^{(a+p)} \mod N \\ &= (g^b)^{a+p} \mod N \\ &= g^{b(a+p)} \mod N \quad \text{-By power rules} \end{aligned} \tag{39}$$

And the server calculates:

$$\begin{aligned} &(Av)^b \mod N \\ &= (g^a v)^b \mod N \\ &= (g^a \times g^p)^b \mod N \quad \text{-By definition of v} \\ &= (g^{a+p})^b \mod N \quad \text{-By power rules} \\ &= g^{b(a+p)} \mod N \quad \text{-By power rules} \end{aligned} \tag{40}$$

Clearly, K_{server} and K_{client} are the same, as required. □

b. We know Mallory knows $v \equiv g^p \mod N$ and can choose A, and therefore knows $g^a \mod N$. Using this information, she can choose A in such a way that she will always know what the key will be. The process would be as follows:

   (i) Step 1: Pick a value A such that A is the modular inverse of v. We know that v has an inverse, since N is prime, and all other integers are co-prime to N; that is the greatest common divisor between N and any integer is 1. From MATH 271, we know we can use the Euclidean algorithm to find this inverse. Mallory then sends the tuple *(I,A)* to the server, using the I that Mallory obtained.

   (ii) Step 2: On the server side, the server computes $g^b \mod N$ using a randomly generated value b and sends this value B back to Mallory.

   (iii) Step 3: The server will now compute $(Av)^b \mod N$ as per the protocol. Well, since A is the modular inverse of v, $(Av) = 1$. The server will just be calculating $1^b \mod N$ which will always be 1. Mallory can essentially disregard the value B she receives, as she knows that K_{server} will always be 1. So, Mallory will always generate $K_{client} = 1$ and she will always be correct, so long as she follows this procedure.

c. Suppose an attacker can solve the key recovery problem. This means that from A, B, and v, the attacker can solve either $g^{b^{a+p}} \mod p$ or $(Av)^b \mod p$. Since it is assumed the attacker does not know a or b, they must have some way to solve for these values. This is essentially the discrete logarithm problem, that is given $g^a \mod p$ where p is a prime and g is a prime root of p, solve for a. As discussed in class, the security of the Diffie-Hellman protocol relies on the discrete logarithm problem being computationally hard, since it is the equivalent of calculating the pre-image of a one way function. However, if someone can solve this key recovery problem, they must be able to solve the discrete logarithm problem, and by solving this problem they would be able to solve the Diffie-Hellman problem.