

## Question 1

A) My UB number as a 32-bit vector by converting each digit into a 4 bit binary number

Ub number: 18012352-bit vector

Scale to calculate this binary to digit conversion

	8	4	2	1
1	0	0	0	1
8	1	0	0	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
5	0	1	0	1
2	0	0	1	0

1 = 0001 , 8= 1000 , 0= 0000, 1= 0000, 2= 0010 , 3= 0011, 5= 0101 , 2 = 0010

B)

Key = 010011


Iv vector = 001011

The way I have done this is taking the Original iv vector – 001011 and doing the permutation of m4 m6 m5 m2 m1 m3 then with that new iv vector will xor this with the key = 010011 then take the first 4 bits of that new outcome and xor that with my actual ub number bits then going onto the next step I will take the next iv vector that I had permuted and xored with the key and then xor that with the original key and then repeat this step until the last stage of the encryption as I will show below.


Stage 1 –

001011 – original lv 

011001- permuted new lv

010011 – key 

001010 – new permuted iv xored with the key

 0001 – my original message first bit

0011 – this is the first 4 bits in red xored with my original message first bit


Stage 2 –

So, the new iv vector will be used from the last step basically this is – 001010


And this will be permuted and then xored with the key and then the same process as previous

001010 – lv 

001001- permuted new lv

010011 – key 

011010 – new permuted iv xored with the key

 1000 – my original message first bit

1110 – this is the first 4 bits in red xored with my original message first bit


Stage 3 –

So, the new iv vector will be used from the last step basically this is – 011010


And this will be permuted and then xored with the key and then the same process as previous

011010– New iv 

001110- permuted new lv

010011 – key 

011101 – new permuted iv xored with the key

 0000– my original message first bit

0111 – this is the first 4 bits in red xored with my original message first bit

Stage 4 –

So, the new iv vector will be used from the last step basically this is – **011101**

And this will be permuted and then xored with the key and then the same process as previous

011101 – New iv

110101- permuted new iv

010011 – key ⊕

**100110** – new permuted iv xored with the key

⊗ 0001 – my original message first bit

1000 – this is the first 4 bits in red xored with my original message first bit

Stage 5 –

So, the new iv vector will be used from the last step basically this is – **100110**

And this will be permuted and then xored with the key and then the same process as previous

100110 – New iv

010011 - permuted new iv

010011 – key ⊕

**111001** – new permuted iv xored with the key

⊗ 0010 – my original message first bit

1100 – this is the first 4 bits in red xored with my original message first bit

Stage 6 –

So, the new iv vector will be used from the last step basically this is – **111001**

And this will be permuted and then xored with the key and then the same process as previous

111001 – New iv

010111- permuted new iv

010011 – key ⊕

**100100** – new permuted iv xored with the key

⊗ 0011 – my original message first bit

1010 – this is the first 4 bits in red xored with my original message first bit

Stage 7–

So, the new iv vector will be used from the last step basically this is – **100100**

And this will be permuted and then xored with the key and then the same process as previous

100100 – New iv

100010- permuted new iv

010011 – key ⊕

**110001** – new permuted iv xored with the key

⊗ 0101 – my original message first bit

1001 – this is the first 4 bits in red xored with my original message first bit

Stage 8 –

So, the new iv vector will be used from the last step basically this is – **110001**

And this will be permuted and then xored with the key and then the same process as previous

110001 – New iv

010110- permuted new iv

010011 – key  $\oplus$

**000101** – new permuted iv xored with the key

$\oplus$  0010 – my original message first bit

0011 – this is the first 4 bits in red xored with my original message first bit

New 32 bit after encryption - 0011 , 1110 , 0111 , 1000 , 1100 , 1010 , 1001 , 0011

## Question 2 -

A) In the Woo and Lam Pi protocol A is claiming the identity and B is verifying the identity claim made by A. In the first step A states its identity to B however at this early stage of the interaction A does not trust B. In the second step B sends a challenge to A as it does not trust it to provide proof of its identity to demonstrate its knowledge of  $K_{AS}$  by encrypting  $N_B$  with it. In the third step of this communication is demonstrating that A has responded to the challenge by encrypting  $N_B$  with  $K_{AS}$  and returning that message to B. In the fourth step B communicates with the server as it does not completely trust A and the message between these two is that B is challenging S to prove its knowledge of  $K_{BS}$  by encrypting the message it has sent ' $\{A, \{N_B\}_{K_{AS}}\}$ ' B doesn't trust S however after this message S trusts B because only B can have knowledge of the message it has just constructed. Alongside this B is asking S to identify A with the key  $K_{AS}$  by decrypting  $N_B$  with  $K_{AS}$ . In the final step in the protocol the server responds back to B and issues a challenge to it to prove that it knows  $K_{BS}$  by decrypting  $N_B$  with  $K_{AS}$  from the third step which was the last direct line between A and B. After this B would decrypt this message and the outcome that it's looking for is to see if the nonce is the same as was originally sent and if this is the case then B can trust the validity of A due to the process of interaction. A/B also knows this to be secure as B interacted and issued a challenge of validity to S which then resulted in the secure message of  $N_B$  with  $K_{BS}$ .

The first Attempt to impersonate A fails at step 3 because steps 1 and 2 are basics issuing challenges however at step 3 when it's time to authenticate, I would fail because it does not know  $K_{AS}$

- (1)  $I(A) \rightarrow B: A$
- (2)  $B \rightarrow I(A): N_B$
- (3)  $I(A) \rightarrow B: \{N_B\}_{K_{AS}}$

The second attempt to impersonate B fails at step 4 because the intruder does not have knowledge of  $K_{BS}$

- (1)  $I(A) \rightarrow B: A$
- (2)  $B \rightarrow I(A): N_B$
- (3)  $I(A) \rightarrow B: \{N_B\}_{K_{AS}}$
- (4)  $I(B) \rightarrow S: \{A, \{N_B\}_{K_{AS}}\}_{K_{BS}}$

The third attempt to impersonate S fails at step 5 as intruder will not have knowledge of  $K_{BS}$

- (1)  $I(A) - B: A$
- (2)  $B - I(A) : Nb$
- (3)  $I(A) \rightarrow B: \{Nb\}_{Kas}$
- (4)  $I(B) \rightarrow S: \{A, \{Nb\}_{Kas}\}_{Kbs}$
- (5)  $I(S) \rightarrow B: \{Nb\}_{Kbs}$

b)

In encryption security it is usually computationally expensive to have unique messages and identifiers and a large message space so designers and security individuals came up with the idea to minimize this by eliminating the identifier if that identity was able to be corrupted by an intruder and reveal their identity which would lead to a naming flaw. The main issue with this of course is that this would leave the identity of the individual at risk of the intruder targeting this and convincing a participant that it is a legitimate individual which would lead to the intruder gaining access to the conversation

The protocol below presents a naming flaw, and the flaw is that A does not identify itself as the originator of  $N_a$  so the intruder can impersonate b

- $(\alpha 1)$   $A \rightarrow B(i) : N_a$
- $(\beta 1)$   $I(B) - A : A : N_a$
- $(\beta 2)$   $A - I(B) : \{N_a, N_b\}_{Kab}$
- $(\alpha 2)$   $B \rightarrow A : \{N_a, N_b\}_{Kab}$
- $(\alpha 3)$   $A \rightarrow (i)B : N_b$
- $(\beta 3)$   $I(B) - A : N_b$