

AES KEY – ENCODED IN THE MACHINE READABLE ZONE OF A EUROPEAN EPASSPORT

ORYX STREAM CIPHER ATTACK - PART 1

Vasudha Venkatesh, Anusha Sridharan

AES KEY – ENCODED IN THE MACHINE READABLE ZONE OF A EUROPEAN EPASSPORT	2
1. PROBLEM STATEMENT	2
2. CRYPTANALYSIS TASK:	3
3. PASSPORT INFORMATION BACKGROUND:	4
3.1 MACHINE READABLE ZONE	4
3.2 CHECK DIGIT CALCULATION	5
4. DECRYPTION:	7
4.1. Kseed CALCULATION:	7
4.2 Kenc Calculation:	8
4.3 Decode the base64 code	8
4.4 Perform decryption:	9
5. OBSERVATIONS ABOUT THE SYSTEM:	9
6. ATTACK ON THE SYSTEM:	10
7. WORK FACTOR FOR THE ATTACK	12
8. Suggestions for new challenge:	13
Oryx stream cipher part 1:	13
1. Overview:	13
2. Cryptanalysis:	14
3. POSSIBLE EXTENSION OF THIS PROBLEM - ORYX stream cipher part 3:	19
4. Conclusion:	22

Unfortunately, during transmission a character was lost and has been highlighted with a ?. Nevertheless, you can make it visible again with the help of [2]. To be able to compute the key KENC afterwards you can find an overview of the applied encoding protocols in [3], [4] and an example in [5]. The AES-encrypted message contains a code word that is to be entered as the solution.

References:

The following documents are available online at:

http : www2.icao.int

[1] ICAO MRTD DOC 9303 Part 1 Vol 1, p. IV-16 (Data structure of the lower machine readable line) and p. IV-42

[2] ICAO MRTD DOC 9303 Part 1 Vol 1, p. IV-24 to IV-26 (Check digits in the machine readable zone)

[3] ICAO MRTD DOC 9303 Part 1 Vol 2, p. IV-13 (MRTD Basic Access Control)

[4] ICAO MRTD DOC 9303 Part 1 Vol 2, p. IV-32

[5] ICAO MRTD DOC 9303 Part 1 Vol 2, p. IV-40 IV-41

Note: There is an error in the problem statement. The MRZ information is

12345678<8<<<1110182<111116?<<<<<<<<<<<<<<<2

2. CRYPTANALYSIS TASK:

We follow the below steps to decrypt the ciphertext.

Step 1: Find the missing digit in the given passport.

Step 2: Find K_{seed} using passport MRZ information.

Step 3: Find K_{enc} which is the AES key.

Step 4: Decode the given ciphertext because it is base64 encoded.

Step 5: Decrypt and get the code word which is the solution.

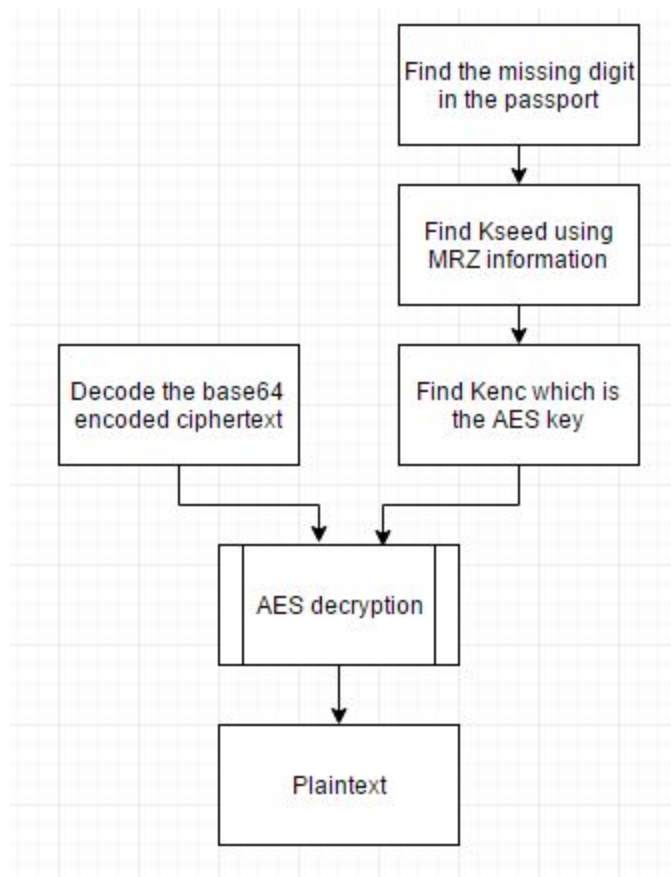


Fig1: Ciphertext decryption steps

3. PASSPORT INFORMATION BACKGROUND:

Before performing the cryptanalysis tasks, we need to understand the significance of each field in the given passport.

3.1 MACHINE READABLE ZONE

The given information is line 2 of the Machine Readable Zone of the passport. It is located in front of the Machine Readable Passport page.

Table 1: Data structure of MRZ - line 2.

MRZ Character Position	Data element	Specifications	Number of characters
1 to 9	Passport Number	9 digit passport number. If there is any unknown digit, use < as filler.	9
10	Check digit	Shall be calculated as specified in section	1

		3.2.1	
11 to 13	Nationality	Nationality of the passport holder. In this problem statement it has been replaced with < which carries a weightage of 0	3
14 to 19	Date of birth	Date of birth of the passport holder in YYMMDD format.	6
20	Check digit	Shall be calculated as specified in section 3.2.2	1
21	Sex	Male, Female or unspecified. In this problem statement it has been replaced with a <.	1
22 to 27	Date of Expiry	Date of Expiry in YYMMDD format.	6
28	Check digit	Shall be calculated as specified in section 3.2.3	1
29 to 42	Personal number or other optional data elements	Any special information by the issuing country. Fill this value till 42nd position with < if there is no special information to fill.	14
43	Check digit	Shall be calculated as specified in section 3.2.4	1

3.2 CHECK DIGIT CALCULATION

The MRZ data has the following information:

- Passport number
- Date of Birth of the holder
- Date of expiry of passport
- Additional information (optional)
- Gender
- Country of issue

A checksum value is computed for each of these fields and appended at the end of each field. These are the 10th, 20th, 28th positions in the MRZ. Finally, a cumulative checksum is calculated and placed in the 43rd position of MRZ.

There is a check digit calculation method used by International civil aviation organization for machine readable travel documents. The steps are as follows:

- Multiply each digit by the corresponding weight.
- Add the products.
- Divide by 10 and calculate the modulus.
- The modulus value which is the remainder is the checksum

For data elements in which the number does not occupy all available character positions, the symbol < shall be used to complete vacant positions and shall be given the value of zero for the purpose of calculating the check digit. Only uppercase alphabetic characters are expected to be in the MRZ text and the alphabets have values 10 to 35 consecutively.

When we do the calculation mentioned above:

Given passport MRZ: 12345678<8<<<1110182<111116?<<<<<<<<<<<<<2

PASSPORT NUMBER:12345678

DATE OF BIRTH: 111018

DATE OF EXPIRY:111116

3.2.1 Check digit calculation - Passport number:

1	2	3	4	5	6	7	8	<
7	3	1	7	3	1	7	3	1

$$7 + 6 + 3 + 28 + 15 + 6 + 49 + 24 = 148$$
$$148 \bmod 10 = 8$$

The tenth digit '8' matches with the checksum we calculated.

3.2.2 Check digit calculation - Date of birth:

1	1	1	0	1	8
7	3	1	7	3	1

$$7 + 3 + 1 + 0 + 3 + 8 = 22$$
$$22 \bmod 10 = 2$$

3.2.3 Check digit calculation - Date of expiry:

1	1	1	1	1	6
7	3	1	7	3	1

= 7+3+1+7+3+6
 =27 mod 10
 =7

So the missing digit should be 7.

3.2.4 Check digit calculation - MRZ

1	2	3	4	5	6	7	8	<	8	<	<
7	3	1	7	3	1	7	3	1	7	3	1

<	1	1	1	0	1	8	2	<	1	1	1
7	3	1	7	3	1	7	3	1	7	3	1

1	1	6	7	<	<	<	<	<	<	<	<
7	3	1	7	3	1	7	3	1	7	3	1

<	<	<	<	<	<	<
7	3	1	7	3	1	7

= 312 % 10
 =2

4. DECRYPTION:

The steps to decrypt the given cipher text is mentioned below:

4.1. K_{seed} CALCULATION:

We use Basic Access Control protocol to construct the MRZ_information which is hashed using SHA1 to calculate K_{seed} value.

- Construct the 'MRZ information' from the MRZ given:
12345678<8<<<1110182<111116?<<<<<<<<<<<<<<<4

Document number = 12345678<

check digit = 3

check digit = 2

check digit = 7

MRZ_information = 12345678<811101821111167

```
Varun@vasu-PC /cygdrive/c/cs265/project1_code
$ ./mrz_attack
The MRZ information that should be used as seed for key calculation is 12345678<
811101821111167
```

2. Calculate the SHA-1 hash of 'MRZ information':

SHA-1 of MRZ=A095F0FDFE51E6AB3BF5C777302C473E7A59BE65

3. Take the most significant 16 bytes to form the Kseed:

k_{seed} = A095F0FD FE51E6AB3BF5C777302C473E

4.2 K_{enc} Calculation:

1. Concatenate Kseed and c:

D=A095F0FDFE51E6AB3BF5C777302C473E00000001

2. Calculate the SHA-1 hash of D:

H_{SHA-1}(D) = EB8645D97FF725A998952AA381C5307909962536

3. Calculate K_{enc} by taking first 128 bits from $H_{SHA-1}(D)$

Kenc=EB8645D97FF725A998952AA381C53079

4. Adjust parity bits

I. Convert the above hexadecimal to binary.

II. If the number of '1's in an octet is odd - do nothing.

III. Else do xor with 00000001 to make the count of 1s odd.

After performing the above steps we get

Kenc=EA8645D97FF725A898942AA280C43179

4.3 Decode the base64 code

The ciphertext given is base-64 encoded message. This has to be decoded using base64 decoder.

Given ciphertext in base64 encoded format:

9MgYwmuPrjiecPMx61O6zluy3MtIXQQ0E59T3xB6u0Gyf1gYs2i3K9Jxaa0zj4gTMazJuApwd6+j
dye15iGHvhQyDHGVIAuYTqJrbFDrfB22Fpil2NfNnWFBTXyf7SDI

After decoding using base64 decoder the ciphertext is given by:

F4c818c26b8fae389e70f331eb53bacc8bb2dccb485d0434139f53df107abb41b27f5818b368b72bd27169ad338f881331acc9b80a7077afa3772788e62187be14320c7195940b984e026b6c50eb7c1db61698a5d8d7cd9d61414d7c9fed20c8

4.4 Perform decryption:

IV=0

AES key = EA8645D97FF725A898942AA280C43179

Ciphertext=F4c818c26b8fae389e70f331eb53bacc8bb2dccb485d0434139f53df107abb41b27f58
18b368b72bd27169ad338f881331acc9b80a7077afa3772788e62187be14320c7195940b984e0
26b6c50eb7c1db61698a5d8d7cd9d61414d7c9fed20c8

After performing decryption:


4865727A6C696368656E20476C7565636B77756E7363682E2053696520686162656E206469
65204E7573732067656B6E61636B742E2044617320436F6465776F7274206C61757465743A
204B727970746F677261706869652101000000000000


Here the value 01000000000000 is the padding value using 01-00 padding scheme.

Converting it to plaintext:



Herzlichen Glueckwunsch. Sie haben die Nuss geknackt. Das Codewort lautet:
Kryptographie!


The code word is the word “Kryptographie”.

**AES key — encoded in the machine readable zone of a European ePassport**
[hick-01] - 46 users already solved this challenge, 3 are working on it.



An AES encrypted message has been forwarded to you. Additionally, you have received the corresponding key - unfortunately not in the form like a machine readable zone on an identity document as it is used e.g. with ePassports in Europe.
[Read more...](#)

 Click [here](#) to get to the corresponding forum topic to share your opinion.
 Click [here](#) to download the challenge.



Congratulations Vasudha Venkatesh! You've already solved this challenge. You are number 45 in the [hall of fame](#) of this challenge.

5. OBSERVATIONS ABOUT THE SYSTEM:

- Weighted sum methodology is used for checksum calculation where the weights of 7,3,1 are used repeatedly and multiplied by the values in the corresponding positions and the sum of their products %10 is used as checksum
- Use of SHA1 to calculate K_{seed} . Unlike a simple substitution cipher where we can figure out statistical information, in this case we can eliminate any relationship between MRZ information and key by using hashing technique which eliminates any connection between the MRZ information and the final key. Hence, if a digit in the MRZ is left out, an attacker cannot glean any information about it from the final ciphertext. (S)he has to compute the MRZ information using random values and recompute the key seed.

- **Key derivation function:** The KDF is used to generate multiple keys from a single seed. In the MRTD document, the keyseed is used to generate 2 keys for 3DES.

The KDF used is SHA1 hash function where we append keyseed to 00000001 to generate a 20 byte key. We truncate it to the first 16 bytes for AES key.

- Parity calculation: This is needed for DES. 64 bit DES is effectively 56 bit DES because of 8 parity bits (one for each byte of DES key). But, in the case of this problem, parity is calculated for AES, reducing the number of effective key bits from 128 to 112 bits.

6. ATTACK ON THE SYSTEM:

An attacker would know how checksum is being calculated in line 2 of MRZ information and can build a system to crack the missing values. The attacker can make use of the check digit calculated at the field level and the check digit calculated for all the data elements and check the two missing digits.

Eg:

Suppose the given MRZ has the month missing in the date of birth like below:

"12345678<8<<<11??182<1111167<<<<<<<<<<<<<<<2"

The attacker will know that the month values range between 01 to 12. He will have just 12 different values to check, which he has to compare with the check digit of that particular field and the check digit of the entire MRZ. This is more like solving a linear equation.

Let the +missing data element be X and Y.

12345678<8<<<11XY182<1111167<<<<<<<<<<<<<2

Calculating the check digit in the method mentioned in section 3.2.2

1	1	X	Y	1	8
7	3	1	7	3	1

$$(7+3+X+7Y+3+8)\%10 = 2 \quad (1)$$

Calculating the check digit of the entire MRZ information:

1	2	3	4	5	6	7	8	<	8	<	<
7	3	1	7	3	1	7	3	1	7	3	1
<	1	1	X	Y	1	8	2	<	1	1	1
7	3	1	7	3	1	7	3	1	7	3	1
1	1	6	7	<	<	<	<	<	<	<	<
7	3	1	7	3	1	7	3	1	7	3	1
<	<	<	<	<	<	<					
7	3	1	7	3	1	7					

$$(305 + 7X + Y) \% 10 = 2 \quad (2)$$

We can solve (1) and (2) equations to get the values of X and Y.

=> X=1 and Y=0

Since modulus is involved in calculating checksum, there are chances of collision to occur. To overcome this, the attacker can follow a simple technique. Since he knows what each data element stand for, he can apply that to verify the range of missing values.

Below are various scenarios tested and their respective outputs:

Month missing in date of birth field

```
anusha@anusha-VirtualBox:~$ gcc crypto2.c -O0 -g
anusha@anusha-VirtualBox:~$ ./a.out

Given MRZ: 12345678<8<<<11??182<1111167<<<<<<<<<<<<<<<2

Missing values are 1 and 0

The MRZ used for Kseed is 12345678<8111018211111672
anusha@anusha-VirtualBox:~$
```

Date missing in date of expiry field:

```
anusha@anusha-VirtualBox:~$ ./a.out

Given MRZ: 12345678<8<<<1110182<1111??7<<<<<<<<<<<<<<<2

Missing values are 1 and 6

The MRZ used for Kseed is 12345678<8111018211111672
anusha@anusha-VirtualBox:~$
```

Year missing in date of birth field:

```
anusha@anusha-VirtualBox:~$ ./a.out

Given MRZ: 12345678<8<<<??10182<1111167<<<<<<<<<<<<<<<2

Missing values are 1 and 1

The MRZ used for Kseed is 12345678<8111018211111672
anusha@anusha-VirtualBox:~$
```

One digit of date and one digit of month missing:

```

anusha@anusha-VirtualBox:~$ ./a.out
Given MRZ: 12345678<8<<<?1?0182<1111167<<<<<<<<<<<<<<<<2
Missing values are 1 and 1
The MRZ used for Kseed is 12345678<8111018211111672

```

Currently we are handling only missing numbers. We could program to handle missing alphabets as well. In this case the range of X and Y would be from 0-35 instead of 0-9. The values from 10 to 35 would A-Z consecutively as we are expecting only uppercase characters.

7. WORK FACTOR FOR THE ATTACK

There are 12345678<8<<<11??182<1111167<<<<<<<<<<<<<<<2

There are totally 44 characters in the MRZ. If we go for the case where we do not know any of these characters. The work factor can be calculated as follows:

- a. Passport number(1-9): Passport is 9 characters long and the 10th character is the check digit of the passport number. So the work factor for passport number would be $36^9/2$ as the attacker has to do an exhaustive search. But he could maintain a dictionary of various combinations and check digit calculated for the same.
- b. Nationality(11-13): The next three digit characters represents the nationality of passport holder. The total number of combination the three digits is **196**.
- c. Date of Birth(14-19): Date of birth is of the format YYMMDD. So when the attacker does an exhaustive search, he should search for $10*10*12*31/2 = 18600$ different values. Check digit is calculated based on these values.
- d. Sex(21): **M/F = 2**
- e. Date of Expiry(22-27): This will have same work factor as date of birth.
- f. Other information(29-42): These 14 characters represents the number give by the passport issuing country. This could be alphabet or number of '<'. The attacker can check by substituting '<' in all characters and check for other options if it does not work as many passports have '<' in this field. But if has to do an exhaustive search, it could be $37^{14}/2$. 43rd character is the check digit of this information.



Fig2: Work factor graph showing work factors of 6 fields mentioned above

$$\text{Total work factor} = 36^9 * 196 * 18600 * 2 * 18600 * 37^{14} / 2 \approx 2^{156}$$

Suggestions for new challenge:

As evidenced in the attack explained above, the missing digit's location can be changed to dramatically change the work factor. Instead of removing the checksum value, the missing value could be placed in 2 of the fields mentioned in the previous section. They could remove the cumulative checksum value at the end (to prevent calculations using linear equation) and remove 2 values in the month field. The values could be removed such that there are 2 possible values eg. 12 and 23. In this case, the attacker should have knowledge of the system and know that this is the month field and it cannot have 23 as a possible value and (S)he must choose 12.

Oryx stream cipher part 1:

1. Overview:

The ORYX stream cipher consists of three 32-bit LFSRs X, A, B which are shifted differently depending on some bits in the LFSR X. The key stream is a combination of the highest 8 bits of each of the three LFSRs. It is neither feasible nor necessary to search the whole 96-bit key space, there are more efficient methods!


Given 30 keystream bytes and the L permutation matrix of 256 values, recover the 96 bit key which includes the initial fill of the registers X, A and B concatenated with each other.


Answer:

The initial fill:


$X \lll 1 \parallel A \lll 1 \parallel B \lll 1$ with last bit of $X=1$, last bit of $A=0$, last bit of $B=1$:

beefdeadabcdaaaabbbbabcd


**ORYX Stream Cipher Part I**
[stamp-01] - 67 users already solved this challenge, 8 are working on it.





The ORYX stream cipher consists of three 32-bit LFSRs X, A, B which are shifted differently depending on combination of the highest 8 bits of each of the three LFSRs. It is neither feasible nor necessary to search efficient methods!




Hint: Enter the codeword in hex with non-capital letters.
[Read more...](#)

 Click [here](#) to get to the corresponding forum topic to share your opinion.

 Click [here](#) to download the challenge.

 Click [here](#) to download the additional file of the challenge.



Yes! This was the correct solution. You now will be added to the [Challenge Hall-of-Fame](#) as number 68.

2. Cryptanalysis:

Reference: Applied cryptanalysis: Breaking Ciphers in the Real World, by Mark Stamp and Richard M.Low

All 3 registers X, A and B are filled with initial seed value(which we have to find in this problem). These registers are stepped and then a key byte is generated. Register A and X always step only once. However, register B could step once or twice depending on the 26th bit from the leftmost position of register X where the count starts from 0. After stepping, if the 26th bit is 0, register B steps once. If the 26th bit is 1 then register B steps twice.

A keystream byte K_i is generated using the formula,

$$K_i = (H(X) + L[H(A)] + L[H(B)]) \bmod 256$$

The value $H()$ refers to the high bytes. These are the bits from 24th to 31st position. Programmatically, this could be the lowest 8 bits in 32 bit value.

$L[]$ is the corresponding value of $H()$ in the permutation matrix. In this problem statement, the attacker is assumed to know 30 bytes of keystream and the complete L matrix.

The following steps are followed to figure out the initial fills:

- Guess 8 bits of A and 8 bit of B which are the high bits H(A) and H(B). Together, this is $(2^8) * (2^8) = 2^{16}$ guesses. We are concerned with high bits because they are the only bits used in the keystream byte calculation.
- We compute putative $H(X) = (K0 - L[H(A)] - L[H(B)]) \bmod 256$. This is possible because we know the value of K0. Thus, we have a putative fill H(A), H(B) and H(X).
- Now the registers step and generate K1. At this point there will be an incoming bit or 2 incoming bits. We try to guess the values of the incoming bits. There are 12 possible values for the incoming bits for the registers.

Shift A	Shift B	Possible incoming bit for A	Possible incoming bit(s) for B
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1
1	2	1	00
1	2	1	01
1	2	1	10
1	2	1	11
1	2	0	00
1	2	0	01
1	2	0	10
1	2	0	11

We simulate each of the above 12 scenarios to find all possible ways the registers can step. For each stepping, we calculate next_H(A) and next_H(B). The H(X) from previous step can either have 0 or 1 as incoming bits. We calculate the possible values it can take as next_possible_0_incomingH(X) and next_possible_1_incomingH(X) respectively.

Calculate $\text{next_H}(X) = (K1 - L[\text{next_H}(A)] - L[\text{next_H}(B)]) \bmod 256$.

If $\text{next_H}(X) == \text{next_possible_0_incomingH}(X)$

OR

$\text{next_H}(X) == \text{next_possible_1_incomingH}(X)$

Authors: Vasudha Venkatesh, Anusha Sridharan

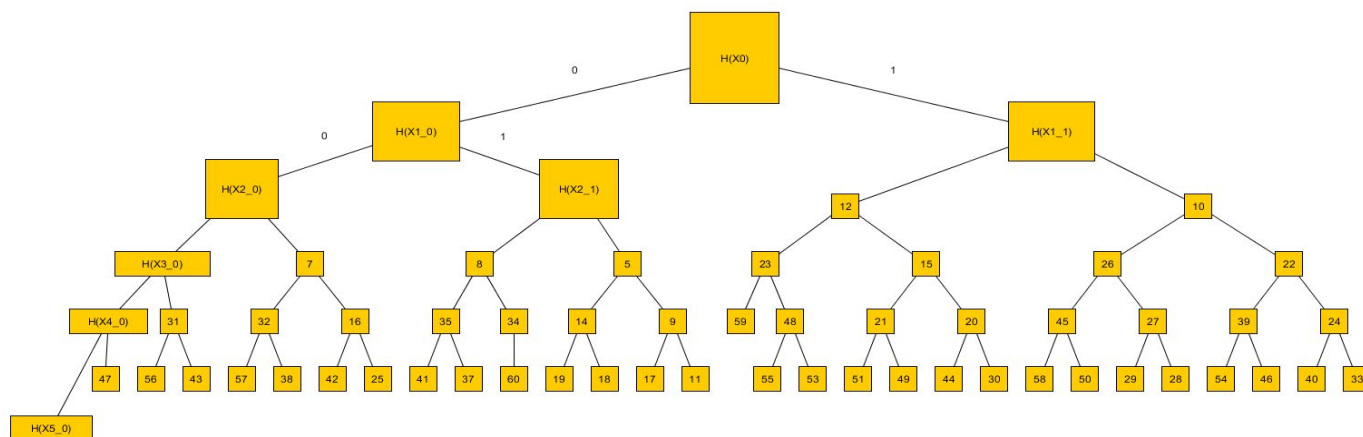
vasudha.venkatesh@sjsu.edu, anusha.sridharan@sjsu.edu

Then, the assumption of putative values of $H(A)$, $H(B)$ and $H(X)$ for generating K_0 is valid enough to generate $\text{next_H}(A)$, $\text{next_H}(B)$ and $\text{next_H}(C)$ for K_1 . we continue our search for valid next step and also keep note of the iteration number from the table above, in order to backtrack and figure out the complete fill of the registers.

How many putative fills are needed?

The search for the valid fills A, B, X can be considered to be a graph traversal where we search for the existence of a connection from $H(X)$ to $\text{next_H}(X)$.

The graph below visualizes the number of putative $H(X)$ values for each of the 2^{16} $H(A)H(B)$ guesses needed to break ORYX cipher when 30 keystream bytes are known. Each key could have 2 different values for $H(X)$ - Extended with 0 or extended with 1.

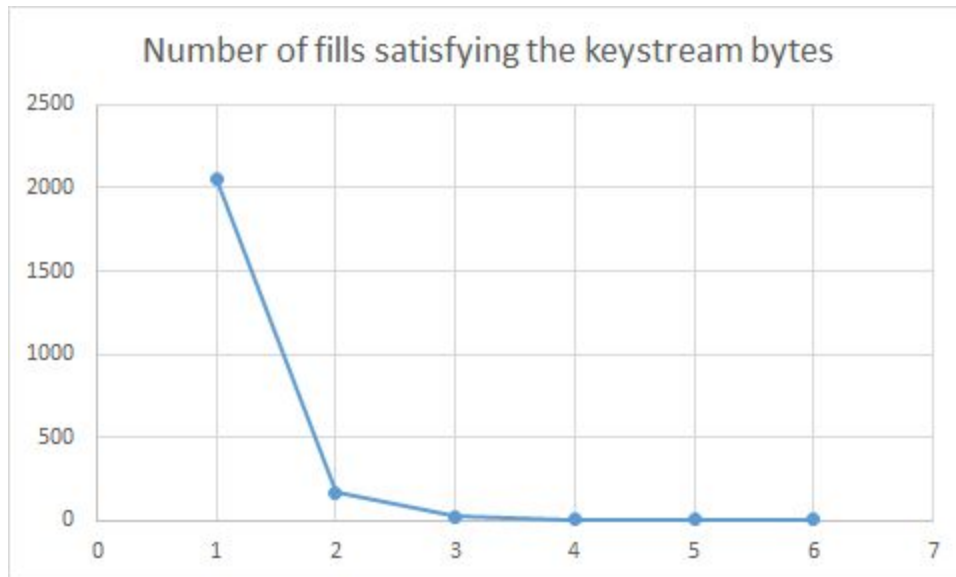


Here, $H(X_0)$ is the value of $H(X)$ assumed to be the state of the register X when K_0 is generated. After stepping, it can have either 0 or 1 as incoming bit. For example, $H(X1_0)$ represents the value of $H(X)$ after the register X steps and has 0 as the next incoming bit and so on. We can infer from the graph that we only need to search depth first with 6 values of $H(X)$ to find the correct combination of $H(A)$, $H(B)$ and $H(X)$ used to generate the keystream.

The chart below was generated from running the program. It generated this data:

1	2057
2	168
3	22
4	5

5	2
6	1



Our program proves that we only need 6 keystream bytes to zero in on one fill value. But, the registers are 32 bits in size. After 6 keystream bytes we only get the high value which is 8 bits. So, after 25 bytes of keystream all the bits in the registers can be found. This is **not** the initial seed. We must step back to generate initial seed.

How to generate initial seed?

The oryx attack code generates the following register fills. These are the fills after the initial seed is shifted. This fill below is used to generate K0.

A

0 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1

B

0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0

X

0 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0

This should be left shifted either as $(A \ll 1, X \ll 1, B \ll 1)$ or $(A \ll 1, X \ll 1, B \ll 2)$.

When they are left shifted, the rightmost bit is left unknown. Consider the case where the register B is left shifted once.

$A \ll 1, B \ll 1$ and $X \ll 1$

The rightmost bit in A could be 0 or 1

The rightmost bit in B could be 0 or 1

The rightmost bit in X could be 0 or 1

The possible combination of bits between X,A,B are

000

001

010

011

100

101

110

111

There are 8 such possibilities

If B register is left shifted by 2 bits, then the possible combination of bits between X,A,B are:

0000, 0001 and so on. Hence, **there are totally $2 \times 2 \times 4 = 16$ possibilities.**

Determine the shift needed for B using the 26th bit from the left. If it is 0, B is stepped once. If it is 1, B is stepped twice. In our case, it was found that B shifts only once. Hence, it generated 8 possible values for the initial seed value of the registers. We tested all the values against mysterytwisters solutions box to zero in on the final result.

The output is given below:

```

Varun@vasu-PC /cygdrive/c/cs265/project1_code
$ gcc oryx_attack.c -o oryx_attack

Varun@vasu-PC /cygdrive/c/cs265/project1_code
$ ./oryx_attack
A just after initial seed = 55e6d555
B just after initial seed = 5ddd5e6
X after initial seed = 5f77ef56
X <<1, A<<1, B<<1 with 000 = beefdeacabcbdaaaabbbbabcc
X <<1, A<<1, B<<1 with 001 = beefdeacabcbdaaaabbbbabcc
X <<1, A<<1, B<<1 with 010 = beefdeacabcbdaaaabbbbabcc
X <<1, A<<1, B<<1 with 011 = beefdeacabcbdaaaabbbbabcc
X <<1, A<<1, B<<1 with 100 = beefdeadabcbdaaaabbbbabcc
X <<1, A<<1, B<<1 with 101 = beefdeadabcbdaaaabbbbabcc
X <<1, A<<1, B<<1 with 110 = beefdeadabcbdaaaabbbbabcc
X <<1, A<<1, B<<1 with 111 = beefdeadabcbdaaaabbbbabcc

```

What makes ORYX cipher vulnerable?

A stream cipher is only as strong as its linear complexity. In this case, the linear complexity of the ORYX cipher is very low. According to Berlekamp-Massey algorithm, if L is the linear complexity of the stream cipher, we only need $2L$ bits of the shift register to determine the entire keystream.

3. POSSIBLE EXTENSION OF THIS PROBLEM - ORYX stream cipher part 3:

In the previous scenario, the L matrix with all 256 values are completely known. We used putative $H(A)$, $H(B)$ and $H(X)$ to find solution with just 25 keys. But, if the permutation matrix is incomplete and we do not know the $H(B)$ values but we have knowledge of the initial fill of the registers A and X . We also have knowledge of the $L[H(B)]$ values of the first 50 keystream bytes.

The first 50 keystream bytes are:

0x3f,0xff,0xdc,0x91,0xcd,0x06,0xff,0x5f,0x44,0x7d,0x83,0x5a,0x96,0x4c,0x44,0x2d,0xc8,0xf3,0x22,0x9e,0x4f,0xa1,0x21,0x1d,0xd1,0x8e,0xa4,0xe2,0x1f,0x76,0xda,0x12,0xba,0x68,0xa9,0x13,0xe1,0x87,0x12,0x7c,0x40,0x57,0xc7,0x89,0x84,0xf3,0xa0,0xb5,0x08,0x01

The first 50 $L[H(B)]$ values are:

0xb4,0x5d,0xdf,0x68,0x8b,0xc6,0x5b,0x7c,0x8f,0x0a,0xb1,0xf9,0x8f,0xfb,0xb1,0xa6,0x2e,0xfa,0x0a,0x12,0x11,0x07,0x4e,0x5a,0x2f,0x3e,0xdd,0x21,0xe3,0x2d,0x73,0xd7,0x1a,0x2a,0x01,0x1c,0x26,0xfd,0x0e,0x53,0x43,0xf0,0x95,0x8e,0x3d,0xce,0xad,0xed,0xce,0x9f

We have to find as many $L[]$ values as possible and the initial fill of register B .

Best case scenario to maximize the number of $L[]$ values:

A keystream byte K_i is generated using the formula,

$$K_i = (H(X) + L[H(A)] + L[H(B)]) \bmod 256.$$

If for 50 keystream bytes, each $H(A)$ and $H(B)$ pair is unique, then with 2 L values per keystream byte, we are expected to find 100 $L[]$ values. But, this is the best case scenario.

Worst case scenario:

If $L[H(A)]$ yields 50 unique values and if $L[H(B)]$ also happen to be within the set of $L[H(A)]$, we can only find 50 L values totally.

How to determine initial fill of register B:

As we discussed in the previous problem, we only need 6 keystream bytes to zero in on the high byte value $H(B)$ for register B.

We used the following algorithm:

- Step register X.
- Find out the value of 29th bit from left (indexing starts from 0) and use appropriate polynomial to step register A once.
- If the 26th bit of stepped X register is 0, record that register B needs to step once. Else, it steps twice.
- Calculate $L[H(A)] = (K_i - H(X) - L[H(B)]) \bmod 256$ since we know all the values except $L[H(A)]$.
- We also know the value of $H(A)$ because register A's initial fill is known. Thus, we know the mapping between $H(A) \rightarrow L[H(A)]$. We record this mapping too. The output from this is shown below. This output shows decimal values instead of hexadecimal for indexes for the purpose of understanding.

1	1	2	2	2	1	1	2	1	1	2	2	2	2	1	2	2	1	1	2	2	2	2
1	1	1	2	2	1	2	2	1	1	1	2	2	1	2	2	1	2	2	2	2	2	2
1	1	1	1	1	1	2	2	1	1	1	2	2	1	2	2	1	2	2	2	2	2	2
index	L_value																					
119	f8																					
187	d9																					
221	19																					
238	b7																					
247	09																					
251	a4																					
125	56																					
190	bc																					
223	22																					
111	aa																					
183	ee																					
219	6f																					
109	8e																					
182	95																					
91	b5																					
173	18																					
86	63																					
171	de																					
213	8b																					
234	c6																					
245	5b																					
122	29																					
189	9b																					
222	a7																					
239	14																					
253	30																					
254	74																					
127	e5																					
191	35																					
59	05																					
29	26																					
14	d3																					
135	16																					
195	4a																					
97	4c																					
176	0f																					
88	70																					
14	37																					
22	fe																					
11	c9																					
133	66																					
66	94																					
161	99																					
80	a3																					
168	17																					

Note that there are only 44 unique values and not 50 values. This rules out the best case scenario.

Comparing this with the $L[H(B)]$ values, we find that, we need to find the placement for the following L matrix values:

b4 5d df 68 7c 8f 0a b1 f9 8f fb a6 2e fa 12 11 07 4e 5a 2f 3e dd 21 e3 2d 73 d7 1a 2a 01 1c 26
fd 0e 53 43 f0 3d ce ad ed 9f

These account for 42 values.

The remaining values map to the existing indexes found using $L[H(A)]$. This reveals an additional clue.

$L(d5) = 8b$ for the 4th keybyte

$L(ea) = c6$ for the 5th keybyte

$L(f5) = 5b$ for the 6th keybyte

$L(b6) = 95$ for the 42nd keybyte

$L(6d) = 8e$ for the 43rd keybyte

All keybyte indexes start at 0.

With this much information and the shifts for B register for each keystream byte, it is easy to find the correct $H(B)$ value for the starting keybyte just immediately after the initial fill.

The total number of L permutation values that we can find is : $44 + 42 = 86$

Authors: Vasudha Venkatesh, Anusha Sridharan

vasudha.venkatesh@sjsu.edu, anusha.sridharan@sjsu.edu

Conclusion:

The cryptanalysis of ORYX stream cipher gave an insight into the vulnerabilities to watch out for in a stream cipher. The cryptanalysis of E-passport based AES key gave an insight into how machine readable travel documents work and how much hashing prevents analysis of correlation between key and ciphertext. It also exposed the dramatic increase in work factor of attack depending on the location of missing digits.