University Of Padova

Information Security - Lab 2 Report

Group Violin

Ayça Begüm Taşçioğlu
Enrico Tiozzo
Egon Galvani
Christian Cattai

Table of Contents

Task 1 3

Task 2 4

Task 3 4

Task 4 4

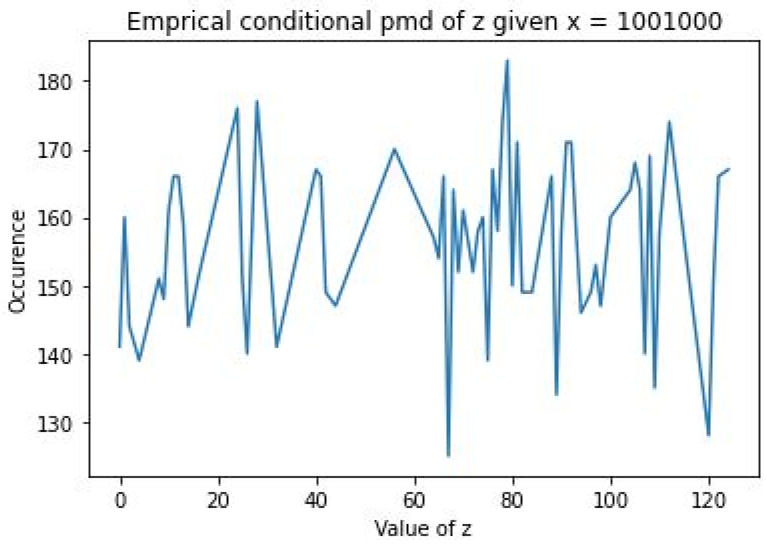
Task 5 8

Task 6 9

Consideration and Remarks 19

Task 1

For task 1 we created a python script, in particular to implement the uniform error wiretap channel we defined the applyMaxDistance function, which given at input x creates a binary string with at most the specified hamming distance from it. The legitimate channel introduces at most 1 binary error per word, while the eavesdropper channel introduces at most 3 binary errors per word. The idea behind the applyMaxDistance function is to: first identify the hamming distance d to be considered, then create a random binary string with d bit at 1, and then calculate the xor between the input and the random binary string generated. At the end of the task, stats_y and stats_z variables are created to store the number of occurrences of every output of eavesdropper and legitimate channels.



Task 2

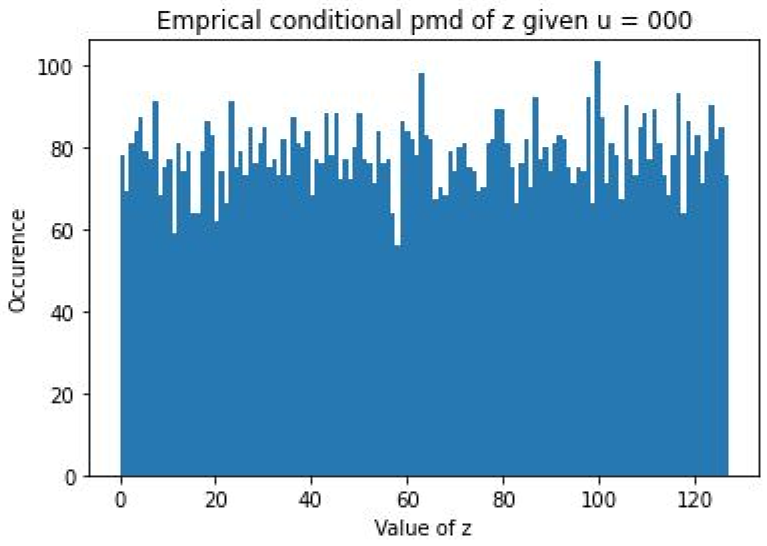
For task 2 we have developed a python script. It gets from keyboard input a three bit string (plaintext), and appends a 0 to it. Then it passes the four bit string to an encoding function that calculates three parity bits according to hamming (7,4). it enqueue these new bits to the plaintext. it then calculates the binary complement and selects a random int from set {0,1}, if the random number is equal to 0 it prints the original ciphertext, otherwise it prints its binary complement.

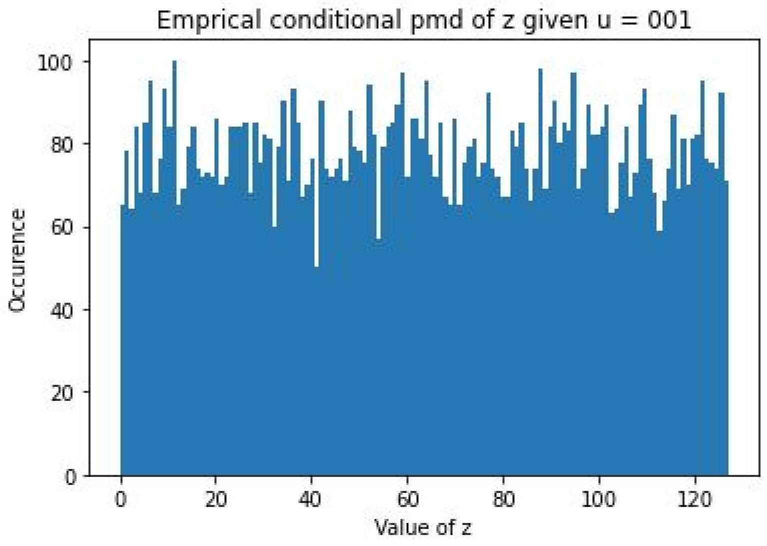
Task 3

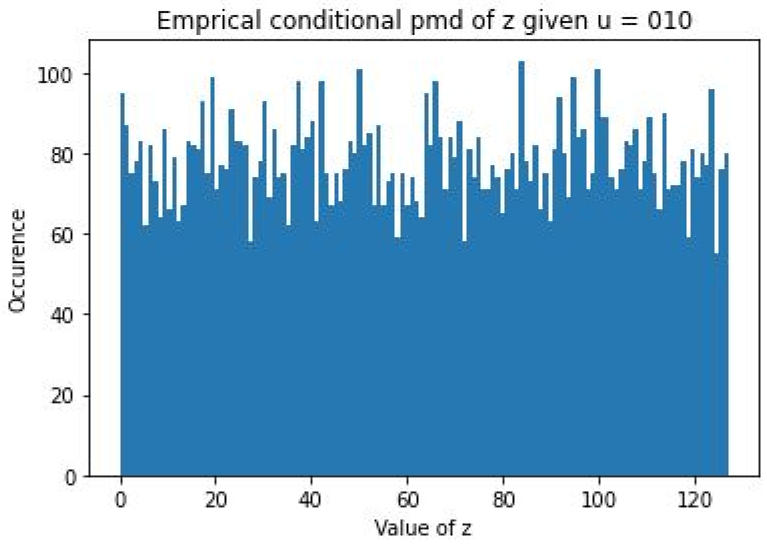
For task 3 we have developed a python script. It gets from keyboard input a seven bit string (ciphertext), it calculates the syndrome, according to hamming (7,4), from the last three bits of the string. Then comparing the value of the sindrome we can understand if the message has an error in one bit or if it is correct. If the syndrome is equal to six there is an error in the first bit (the most significant one), If the syndrome is equal to five there is an error in the second bit, If the syndrome is equal to three there is an error in the third bit, If the syndrome is equal to seven there is an error in the fourth bit. For other values of the syndrome it means that there is no error or it is in the parity bit. It corrects the error if present. Checks if the first bit is equal to 1, if it is, it prints the binary complement of bits 2-4 (considering bits numbered as 1-7), if it is equal to 0 it prints bits 2-4 as they are.

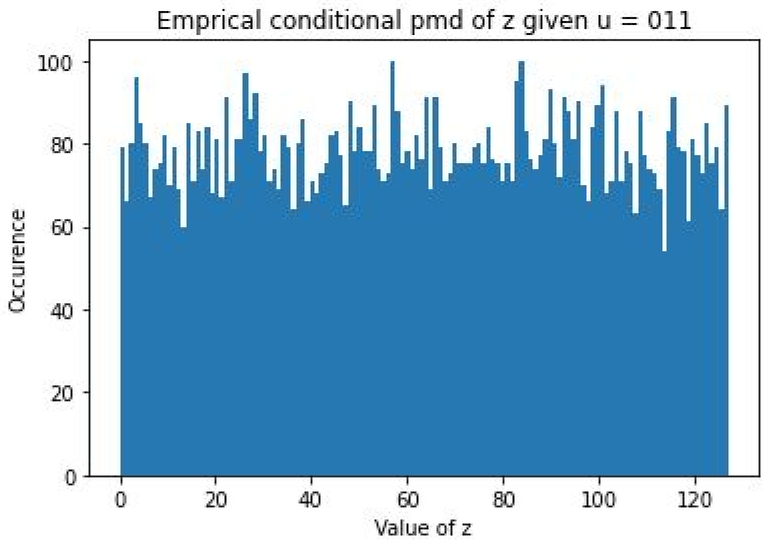
Task 4

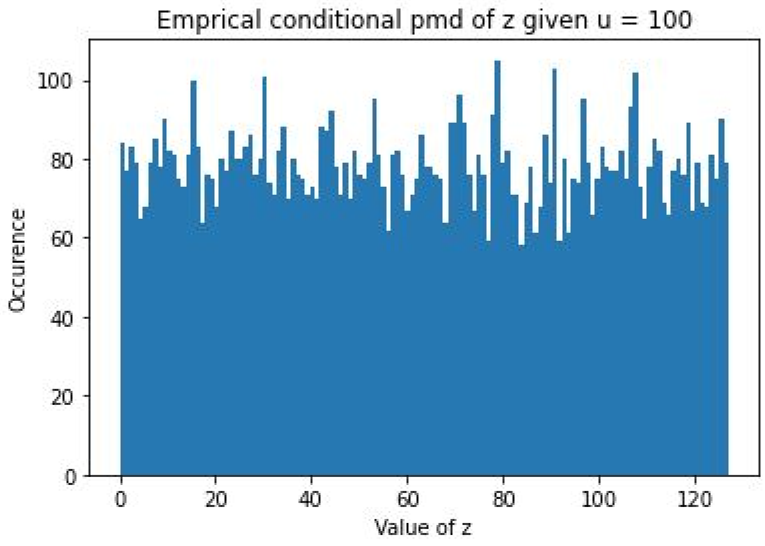
For task4, for each u value that we iterated through a food loop, we created 10**4 realizations of z and compute distribution of z values. We used task2 encode method to encode message u, and reash x; then we used task1 applyMaxDistance method to calculate z values. We kept z values per given u, in a vector then plot the following distribution graphs:

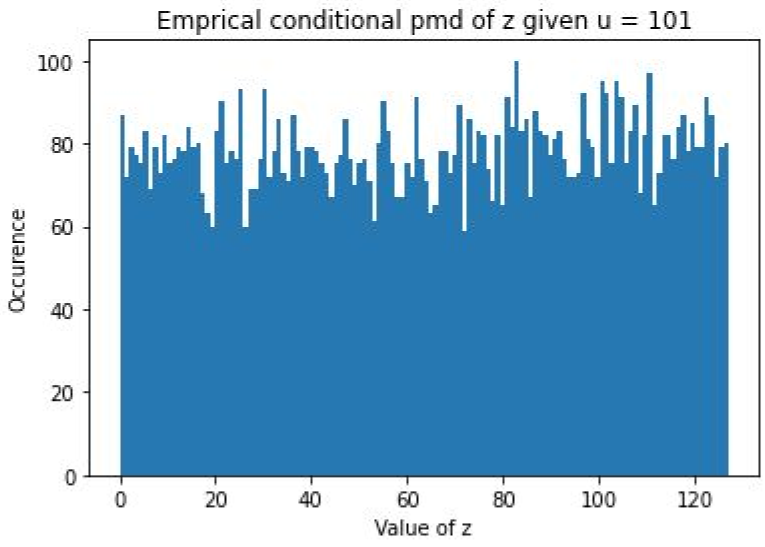

Emprical conditional pmd of z given u = 000
Occurence
Value of z
0
20
40
60
80
100
120

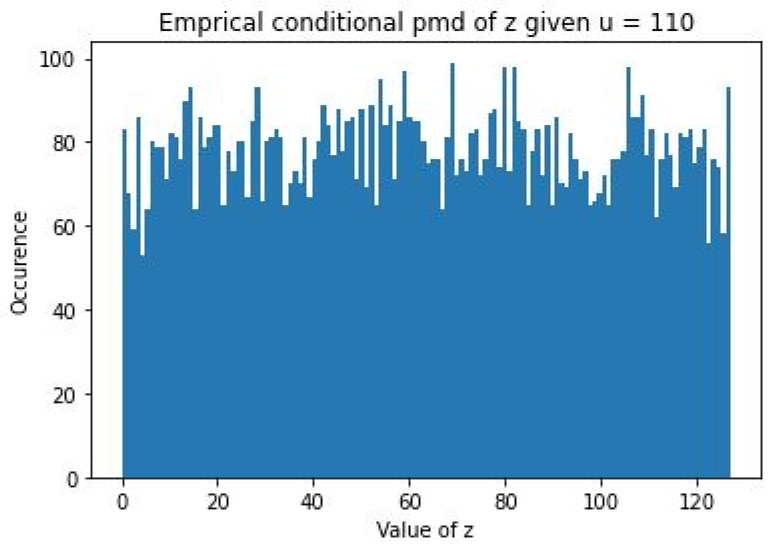

Emprical conditional pmd of z given u = 001
Occurence
Value of z
0
20
40
60
80
100
120

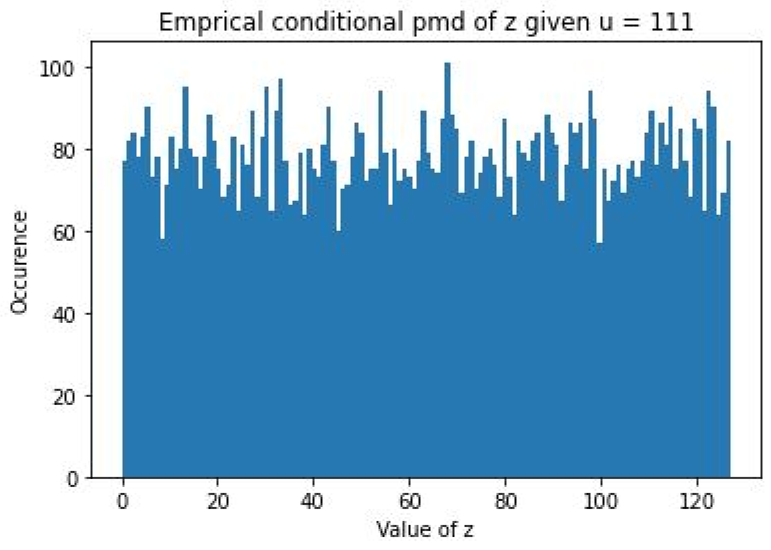

Emprical conditional pmd of z given u = 010
Occurence
Value of z
0
20
40
60
80
100
120


Emprical conditional pmd of z given u = 011
Occurence
Value of z
0
20
40
60
80
100
120

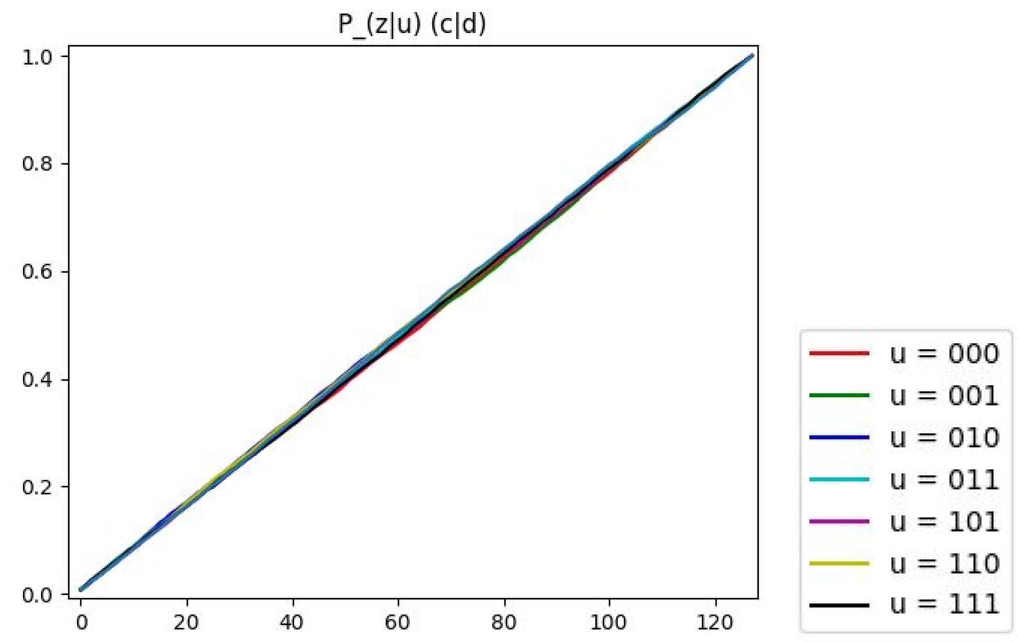

Emprical conditional pmd of z given u = 100
Occurence
Value of z
0
20
40
60
80
100
120


Emprical conditional pmd of z given u = 101
Occurence
Value of z
0
20
40
60
80
100
120





By plotting the pmd for each u = d we can see that is a uniform distribution



Mutual information with u=c,z=d values:

```
Mutual Information I(u,z): 0.007475968361608097
Mutual Information I(u,z): 0.01114477788435242
Mutual Information I(u,z): 0.01226461540712829
Mutual Information I(u,z): 0.008779556357925052
Mutual Information I(u,z): 0.009985790283979648
Mutual Information I(u,z): 0.008184707676020288
Mutual Information I(u,z): 0.010416815728896746
Mutual Information I(u,z): 0.008579674797888398
```

Which are closer to 0, then we can say u and z are empirically independent within the statistical reliability of our simulations.

Task 5

For the task5 we have implemented a binary symmetric channel, with epsilon and delta error probability, respectively for legitimate receiver and eavesdropper channel. In particular we defined the function flipBinStr that allows to flip every bit of the considered binary string with the specified probability. We verified the correctness of the implementation by transmitting a long binary sequence, checking the number of bit errors in each output and estimating from them the value of epsilon and delta used. Then, we connected the wiretap channel with the random binning encoder and decoder from tasks 2 and 3, and simulated several transmissions.

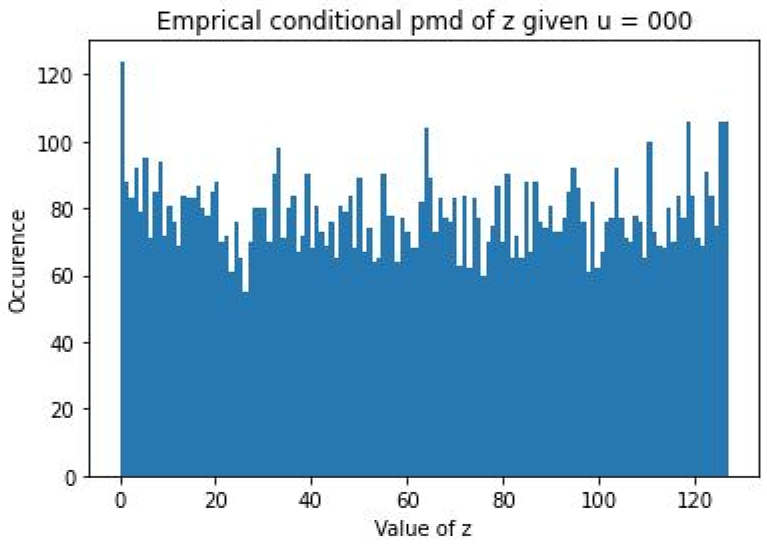
Output of Task5:

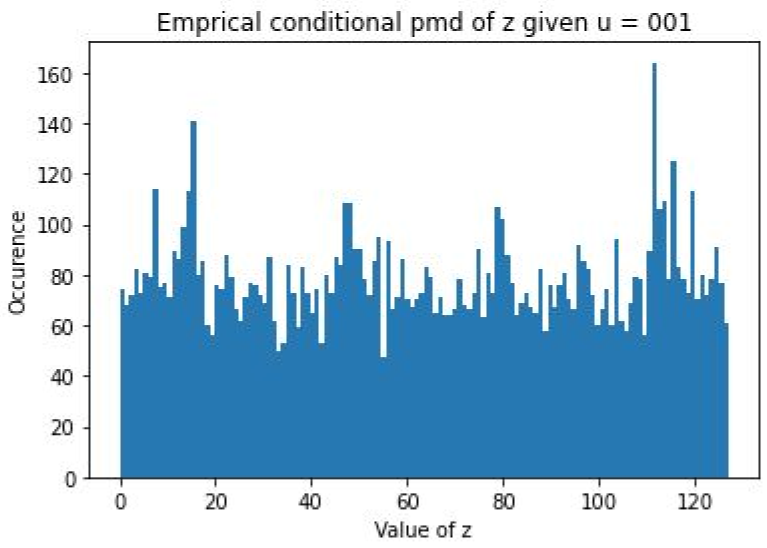
```
Epsilon: 0.200000 - Estimated epsilon: 0.199433
Delta: 0.400000 - Estimated delta: 0.400367
```

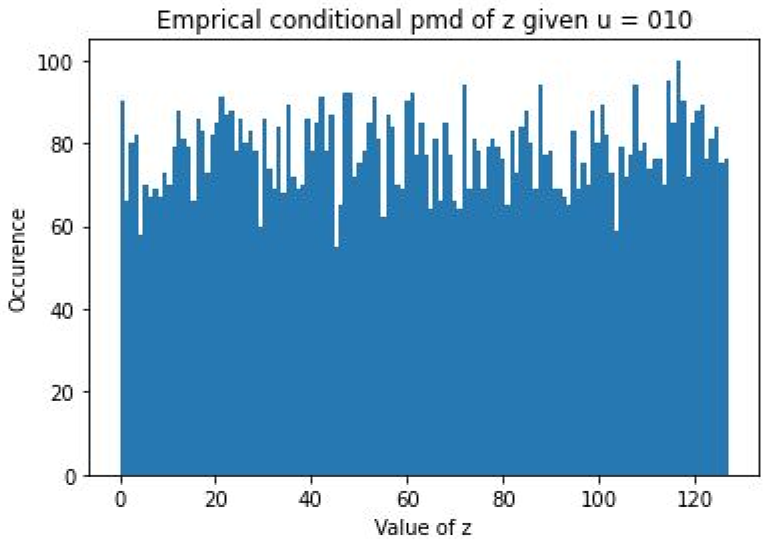
Task 6

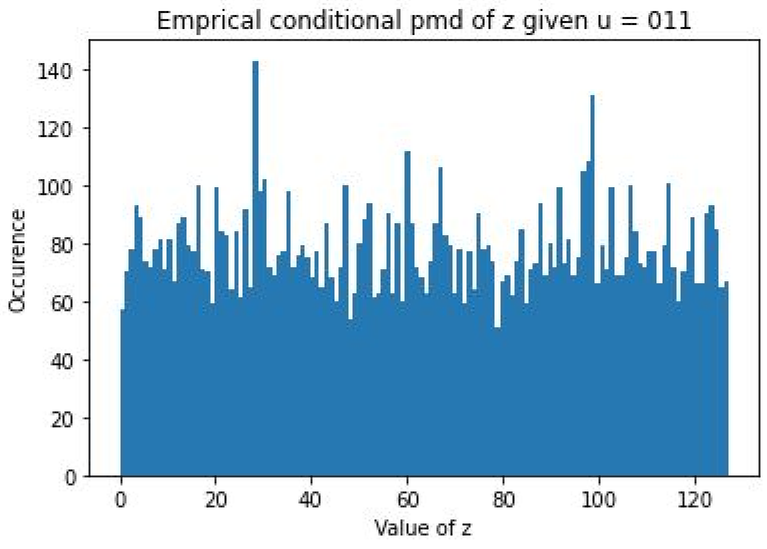
For task6, we first used task2's encode method to encode x with given u values which we iterated over a loop for u values = 0 to 15(2**4 - 1). After that, in a for loop which creates 10**4 realizations, we randomly choose our epsilon and delta values. Based on a randomly chosen epsilon and delta pair, we calculated y and z with the task5's flipBinStr method. Given the below theoretical formulas we derived mutual information:
Mutual Information lim_(n->infinite) I(u;z) = 0
I(u;z) = H(z) - H(z|u)
H(z) = -(sum[p(z)*log_2 p(z)] for all z in Z)
H(z|u) = sum[p(u)*H(z| u=a)] for all u in U
I(u;z) = H(z) + H(u) - H(u,z)
Also I(u;z) can be represented w/same formula in task4

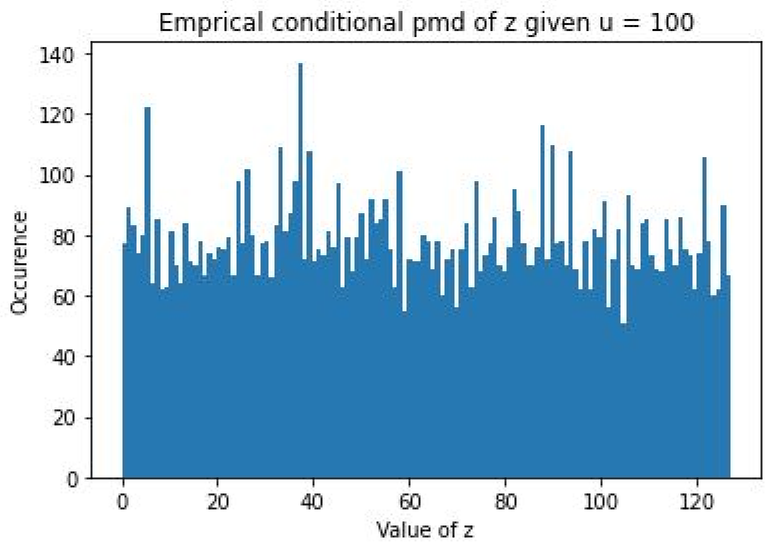
We kept z values in a vector and got following distributions (random epsilon and delta pair):

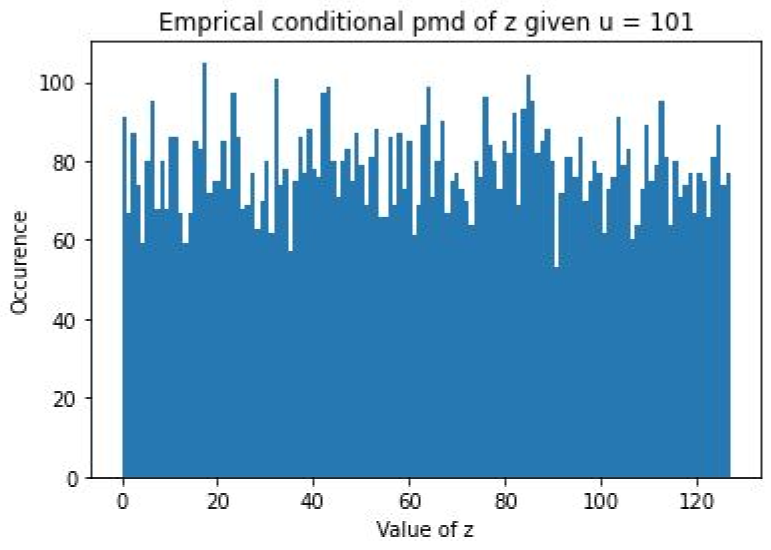


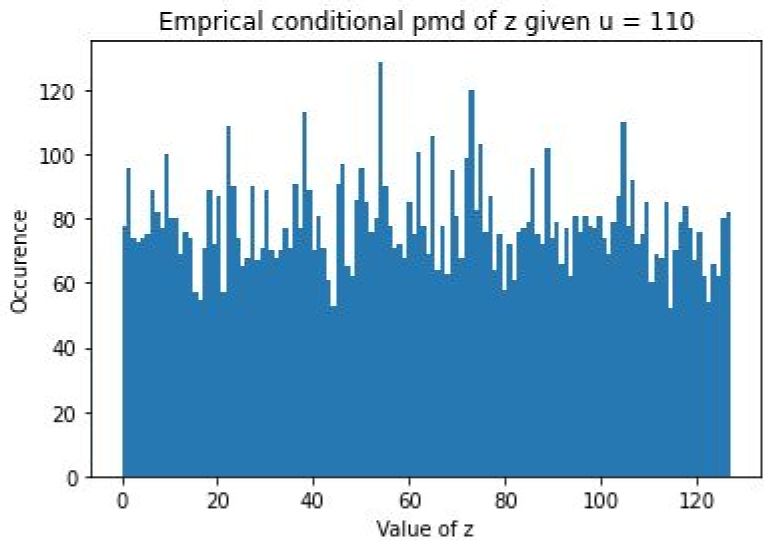

Emprical conditional pmd of z given u = 001
Occurence
Value of z
0
20
40
60
80
100
120
140
160

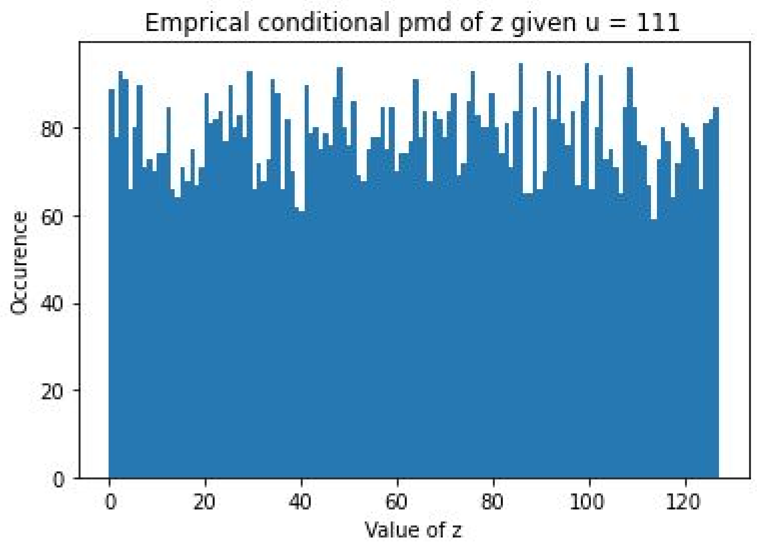

Emprical conditional pmd of z given u = 010
Occurence
Value of z
0
20
40
60
80
100
120


Emprical conditional pmd of z given u = 011
Occurence
Value of z
0
20
40
60
80
100
120
140


Emprical conditional pmd of z given u = 100
Occurence
Value of z

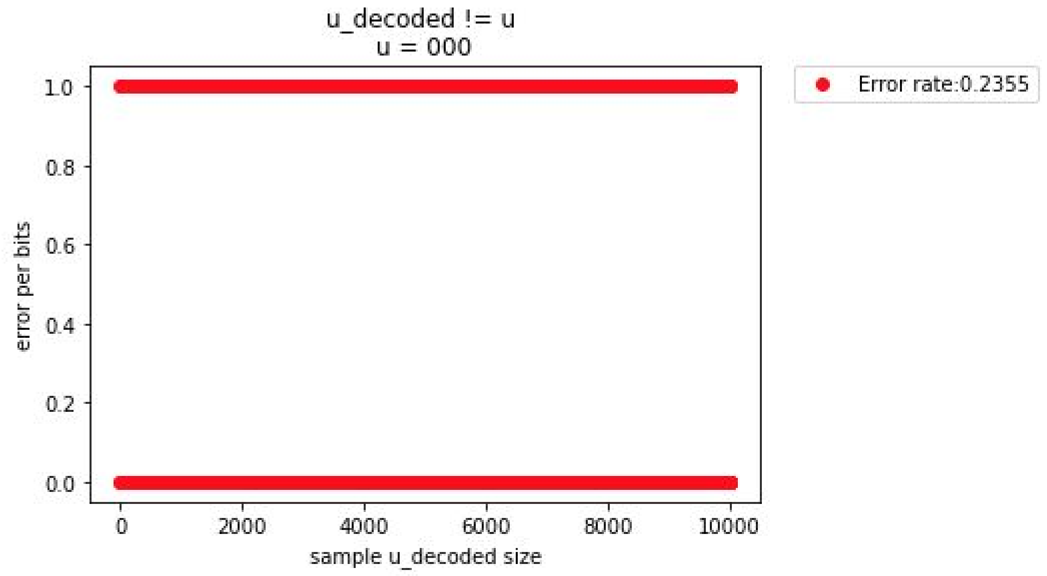

Emprical conditional pmd of z given u = 101
Occurence
Value of z

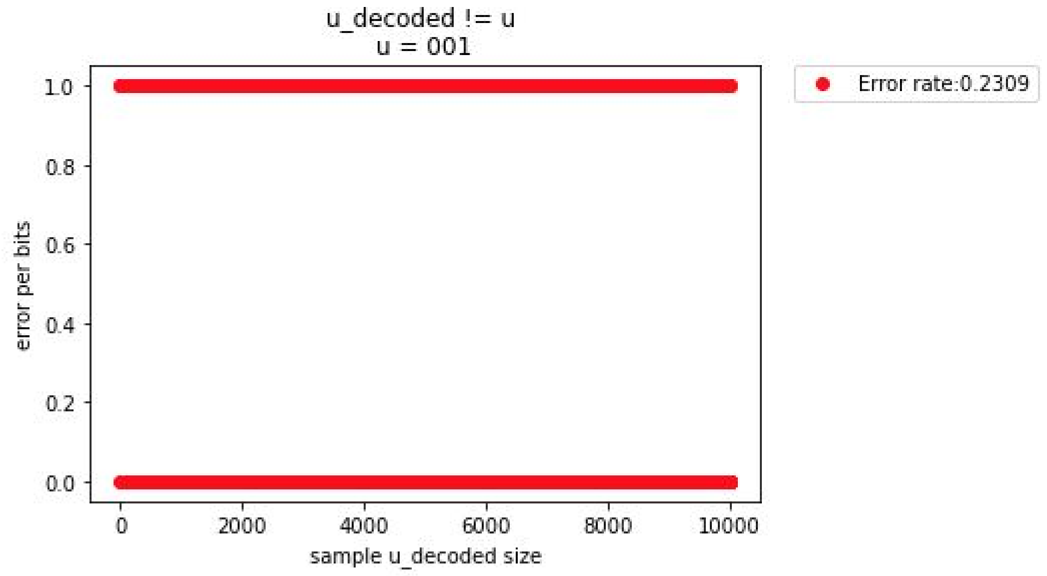

Emprical conditional pmd of z given u = 110
Occurence
Value of z

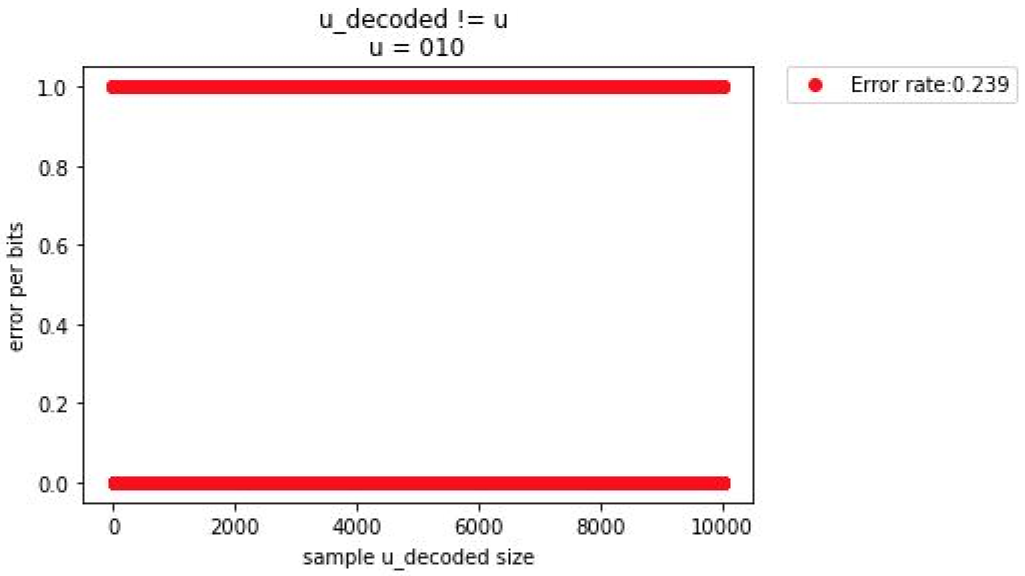


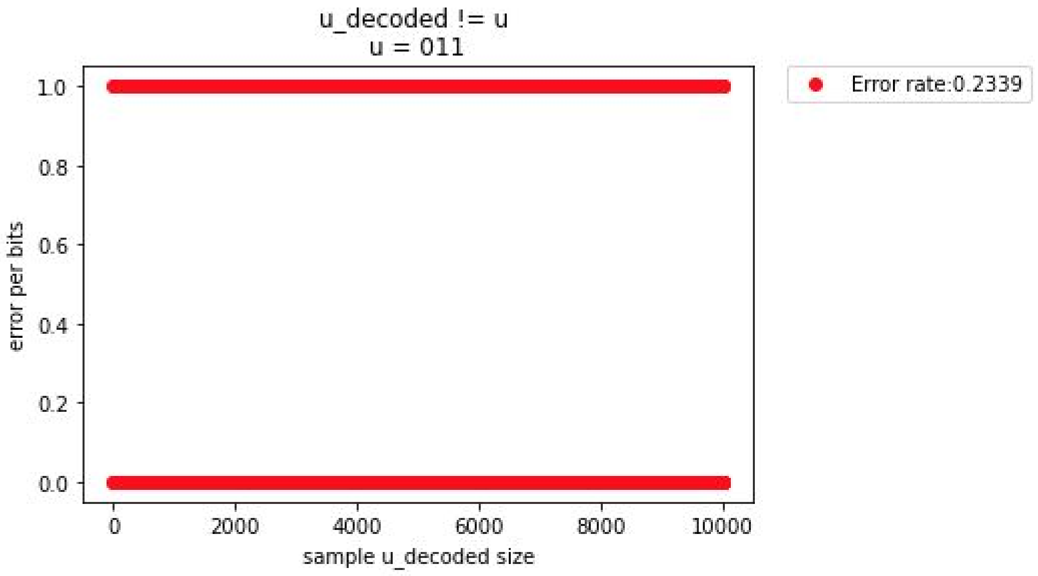
```
Mutual information:
Mutual Information I(u,z): 0.08125246312068969
Mutual Information I(u,z): 0.07466391254515063
Mutual Information I(u,z): 0.07572535058814311
Mutual Information I(u,z): 0.08129016246145737
Mutual Information I(u,z): 0.0834442940507542
Mutual Information I(u,z): 0.0780417065668832
Mutual Information I(u,z): 0.07828426230877404
Mutual Information I(u,z): 0.0751974548308921
```

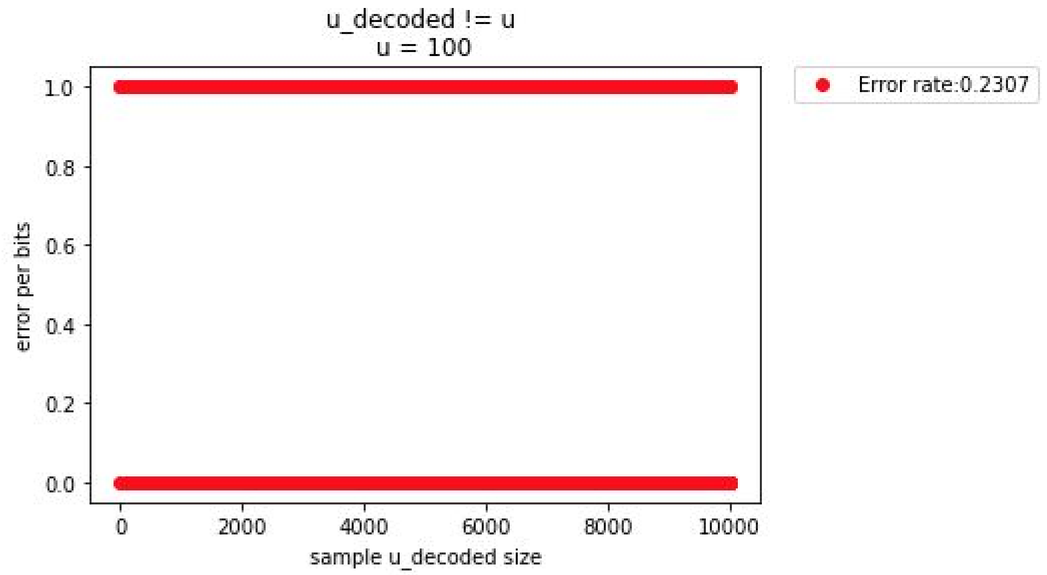
For Bob’s error rate, we created following graphs:

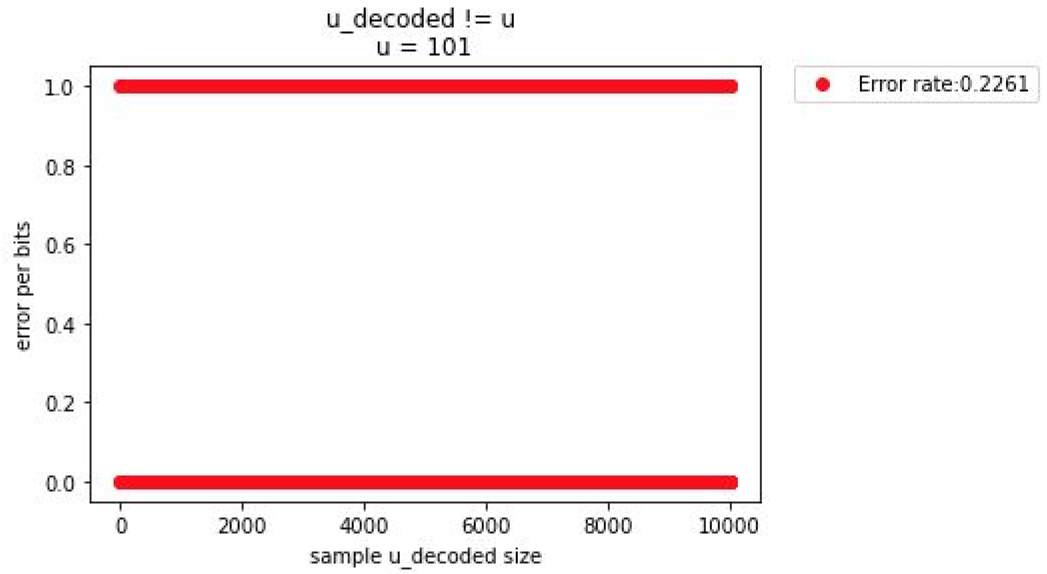


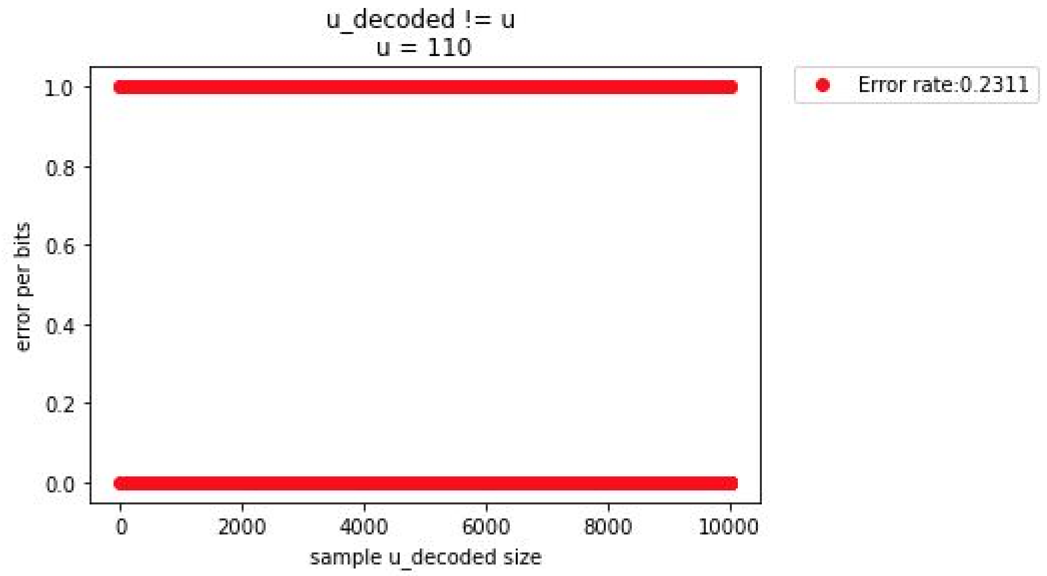


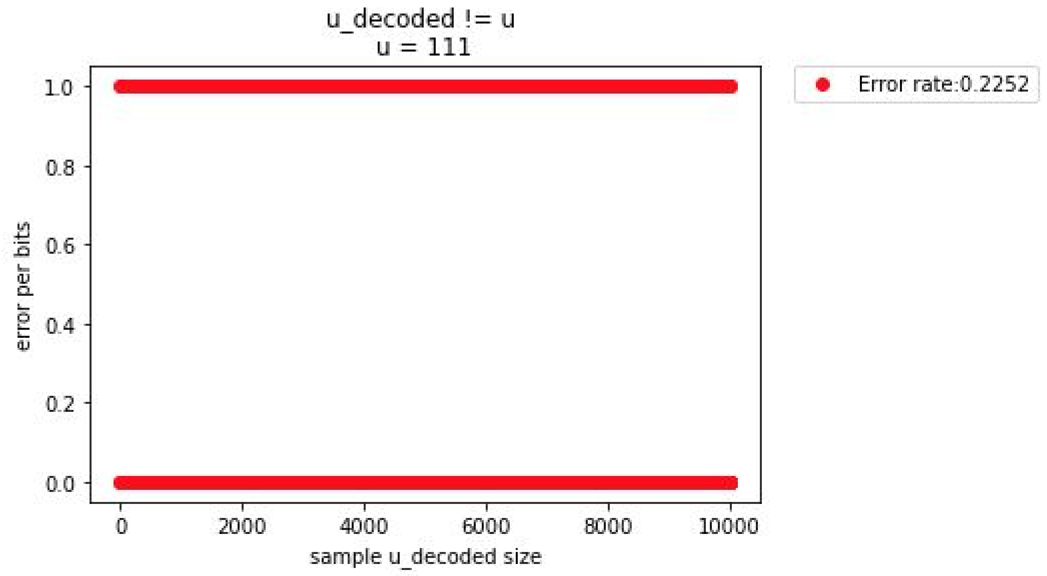

u_decoded != u
u = 010
Error rate:0.239
error per bits
sample u_decoded size
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000


u_decoded != u
u = 011
Error rate:0.2339
error per bits
sample u_decoded size
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000

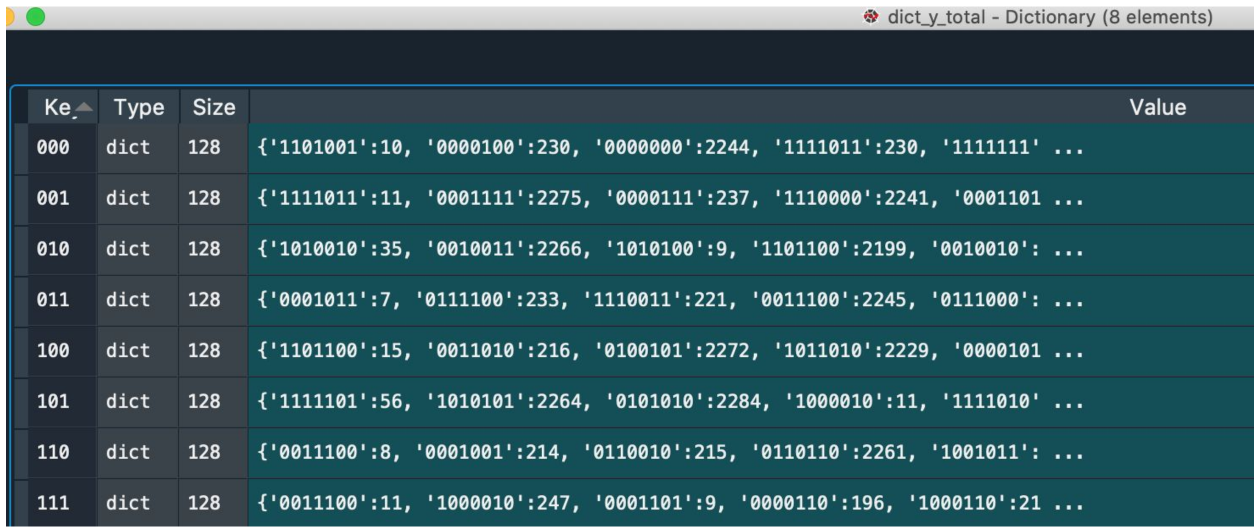

u_decoded != u
u = 100
Error rate:0.2307
error per bits
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
sample u_decoded size


u_decoded != u
u = 101
Error rate:0.2261
error per bits
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
sample u_decoded size


u_decoded != u
u = 110
Error rate:0.2311
error per bits
sample u_decoded size
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000


u_decoded != u
u = 111
Error rate:0.2252
error per bits
sample u_decoded size
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000

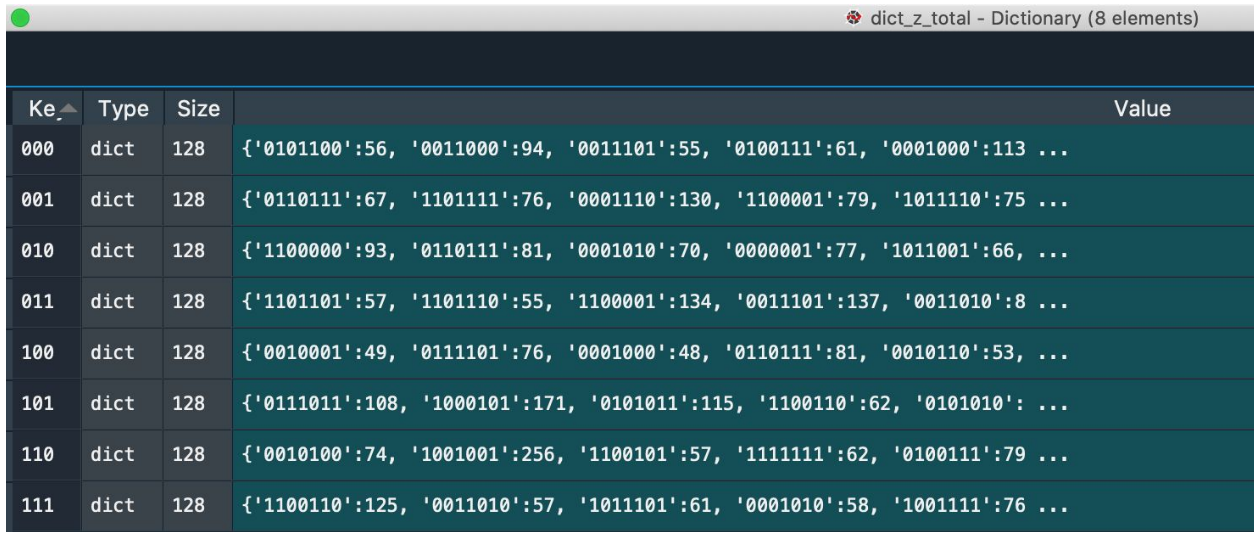
For upper bound analysis, we realized that for each u, we created 10**4 realizations of y and z values, while inspecting these realizations we find that for each u, we can have 128 different y values and 128 different z values:



dict_y_total - Dictionary (8 elements)

Ke	Type	Size	Value
000	dict	128	{'1101001':10, '0000100':230, '0000000':2244, '1111011':230, '1111111' ...
001	dict	128	{'1111011':11, '0001111':2275, '0000111':237, '1110000':2241, '0001101 ...
010	dict	128	{'1010010':35, '0010011':2266, '1010100':9, '1101100':2199, '0010010': ...
011	dict	128	{'0001011':7, '0111100':233, '1110011':221, '0011100':2245, '0111000': ...
100	dict	128	{'1101100':15, '0011010':216, '0100101':2272, '1011010':2229, '0000101 ...
101	dict	128	{'1111101':56, '1010101':2264, '0101010':2284, '1000010':11, '1111010' ...
110	dict	128	{'0011100':8, '0001001':214, '0110010':215, '0110110':2261, '1001011': ...
111	dict	128	{'0011100':11, '1000010':247, '0001101':9, '0000110':196, '1000110':21 ...

Dict_y_total keeps "key" as possible u values, and resulting y values as a dictionary in "value" for given u (y is derived from encoding u and getting x, then implementing wiretap channel). It can be seen that we have
128 different y values = cardinality of Y since y is 7 bit string, |Y| = 2**7 = 128



dict_z_total - Dictionary (8 elements)

Ke	Type	Size	Value
000	dict	128	{'0101100':56, '0011000':94, '0011101':55, '0100111':61, '0001000':113 ...
001	dict	128	{'0110111':67, '1101111':76, '0001110':130, '1100001':79, '1011110':75 ...
010	dict	128	{'1100000':93, '0110111':81, '0001010':70, '0000001':77, '1011001':66, ...
011	dict	128	{'1101101':57, '1101110':55, '1100001':134, '0011101':137, '0011010':8 ...
100	dict	128	{'0010001':49, '0111101':76, '0001000':48, '0110111':81, '0010110':53, ...
101	dict	128	{'0111011':108, '1000101':171, '0101011':115, '1100110':62, '0101010': ...
110	dict	128	{'0010100':74, '1001001':256, '1100101':57, '1111111':62, '0100111':79 ...
111	dict	128	{'1100110':125, '0011010':57, '1011101':61, '0001010':58, '1001111':76 ...

Dict_z_total keeps "key" as possible u values, and resulting z values as a dictionary in "value" for given u (z is derived from encoding u and getting x, then implementing wiretap channel). It can be seen that we have
128 different z values = cardinality of Z since z is 7 bit string, |Z| = 2**7 = 128
In order to find an upper bound estimation properly, we calculated N_(z|x) and N_(y|x) values in our dictionary with given u, given x values. We had 1024 estimations for each u = 3 bit binary string ~ 2^3, and x = 7 bit binary string ~ 2^7, 2^3*2^7 = 2^10

For Secrecy capacity of our wiretap BSC, we calculated Cs with formula:
Cs = C_AB - C_AE = $h_2(\delta) - h_2(\varepsilon)$ where

$h_2(\varepsilon) = \varepsilon \log_{1/2} \varepsilon + (1 - \varepsilon) \log_{1/2}(1 - \varepsilon)$

$h_2(\delta) = \delta \log_{1/2} \delta + (1 - \delta) \log_{1/2}(1 - \delta)$

For epsilon = 0.14
For delta = 0.29
Cs = 0.28448243469654855

For epsilon = 0.13
For delta = 0.42
Cs = 0.4240157100056644

For epsilon = 0.13
For delta = 0.3
Cs = 0.32385271420270356

For epsilon = 0.14
For delta = 0.33
Cs = 0.3306875611368716

For epsilon = 0.17
For delta = 0.39
Cs = 0.3070947697608677

For epsilon = 0.13
For delta = 0.46
Cs = 0.43794025379223667

For epsilon = 0.17
For delta = 0.3
Cs = 0.2235861204864732

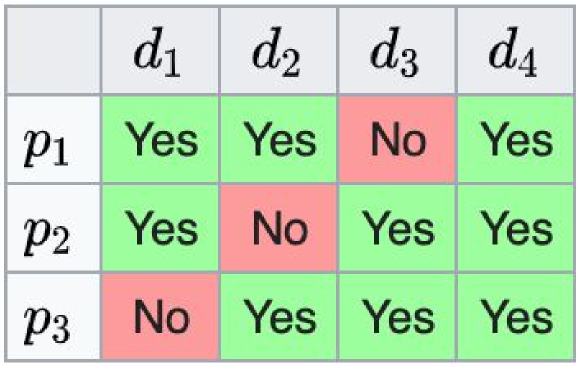
For epsilon = 0.21
For delta = 0.33
Cs = 0.17344363284845388

Consideration and Remarks

1. **How many secret message bits per channel use ("transmitted word") have you obtained with your scheme?**

   How many secret bits per binary digit ("transmitted bit")?

In our encoder, we are encoding a given 3 bit binary message u, with adding 1 leading zero and then completing it to 7 bit binary string with hamming difference 3. We are implementing Hamming(7,4) for encoding u. The Hamming code adds three additional check bits to every four data bits of the message. Parity bits are set as below table



	d_1	d_2	d_3	d_4
p_1	Yes	Yes	No	Yes
p_2	Yes	No	Yes	Yes
p_3	No	Yes	Yes	Yes

x = '0' + u + [p1 mod(2)] + [p2 mod(2)] + [p3 mod(2)]

x_transmitted = x or x_complement with random probability.

E.g given u = "100", we are first adding a lead zero then u = "0100". Then we are setting parity bits and implementing xor operation on them. After that x can be either 7 bit string with hamming distance 3 = '0100101' or the complement of x = 1011010

Probability that [x_transmitted = x] = [x_transmitted = x_complement] ~ 0.5 (implemented in task 2 with 10**4 realizations)

So our ciphertext contains 3 secret bits of u (given len(u) = 3), with ~0.5 probability, it's because P([x_transmitted = x]) = 0.5

2. **Is it possible to obtain 4 secret bits per channel use?**
   If so, how should you change your encoder/decoder? If not, why?
   It can be possible if our u is a 4 bit binary message, we can send it directly to our encoder without adding a leading 0, then we would add parity bits and can obtain a 7 bit ciphertext. Yet in our decoder, we should fix the way we find decoded messages, because in our current system, a 7 bit ciphertext returns a 3 bit plaintext message.

3. **Is it possible to obtain 2 secret bits per channel use?**
 If so, how should you change your encoder/decoder? If not, why?

 Considering 2 bit messages (after adding 2 leading 0’s):

	0	0	0	0
p1	yes	yes	no	yes
p2	yes	no	yes	yes
p3	no	yes	no	yes

Parity1 = 0+0+0 mod 2 = 0

Parity2 = 0+0+0 mod2 = 0

Parity3 = 0+0+0 mod2 = 0

x= 0000000 or x =1111111 (complement)

	0	0	0	1
p1	yes	yes	no	yes
p2	yes	no	yes	yes
p3	no	yes	yes	yes

Parity1 = 0+0+1 mod 2 = 1

Parity2 = 0+0+1 mod2 = 1

Parity3 = 0+0+0 mod2 = 1

x= 0001111 or x = 1110000 (complement)

	0	0	1	0
p1	yes	yes	no	yes
p2	yes	no	yes	yes

p3	no	yes	yes	yes

Parity1 = 0+0+0 mod 2 = 0

Parity2 = 0+0+1 mod2 = 1

Parity3 = 0+0+0 mod2 = 1

x= 0010011 or x = 1101100 (complement)

	0	0	1	1
p1	yes	yes	no	yes
p2	yes	no	yes	yes
p3	no	yes	yes	yes

Parity1 = 0+0+1 mod 2 = 1

Parity2 = 0+1+1 mod2 = 0

Parity3 = 0+1+1 mod2 = 0

x= 0011100 or x = 1100011 (complement)

If we add 2 leading zeros to given 2 bit messages, we can transmit them. For such architecture, we need to modify our encoder in the way it adds 2 leading 0's to the given message u, then implementing Hamming(7 4) to provide x values. For the decoder, we should modify the way we find syndrome.

4. **One could consider evaluating the secrecy of this mechanism by cascading the eavesdropper channel with a decoder and measuring the resulting error rates. What do you expect Eve's error would be? Why resort to (more complicated) evaluating the mutual information?**

It can be seen that, z which is the message transmitted to eavesdropper is uniformly distributed over possible z values(in our case 128 different z values); even for different plaintext message u, occurrence of a particular z values would be similar. For instance, if we say u = 100, z can be "1011011" with probability 0.0157 or z can be "1011011" with probability 0.0153; If u was equal to "101", z can be "1011011" with probability 0.0104. It is hard for eavesdropper to compute plaintext message from received z, because Tz|x is covering all possible z values with similar/close probabilities. By using mutual information, we are showing the independency of z with respect to u values.

Our code can also be found in : https://github.com/aeyc/RandomBiningEncoding