University of Padua

Department of Information Engineering

Information Security Report

Laboratory Session 2

Implementation of random binning encoding and secrecy rate evaluation

Author:
ZANON ALBERTO
MICHELON LUCA
SCREMIN NICOLA
NIKHIL KARAKUCHI CHIDANANDA
PORRO THOMAS

Teacher:
Nicola Laurenti

29 November 2020

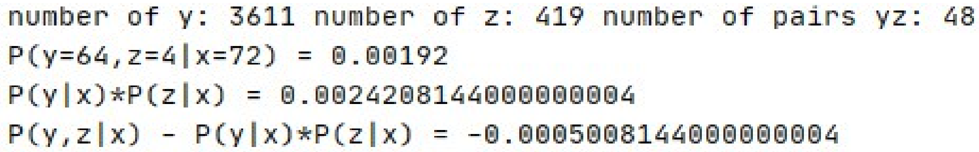
Solution

Our solution to laboratory 2 is entirely implemented using Python. Specifically, we made use of the NumPy library to easily manipulate vectors and quickly compute operations between them. The solution is composed of 7 Python source files: task1.py contains functions necessary to carry out the implement the wiretap channel, so that it accepts an input and produces the corresponding pair of outputs (y; z), task2.py contains random encoder function to implement the random binning encoder, so that it accepts an input and produces the corresponding output, task3.py contain random decoder function to implement the legitimate decoder, so that it accepts an input and produces the corresponding output, task4.py contains the functions implement the encoder + eavesdropper channel, task5.py contains function to implement the wiretap BSC, task6.py contains functions for repeat the simulations in Tasks 3-4 with the wiretap BSC, evaluate the resulting reliability in terms of Bob's error rate on the secret message chain P, evaluate the resulting the secrecy in terms of leaked information to Eve on the secret message I and compute an upper bound to the mechanism security in terms of distinguishability from the ideal counterpart and utils.py contains function to convert the string to array and the array to string.

Task 1

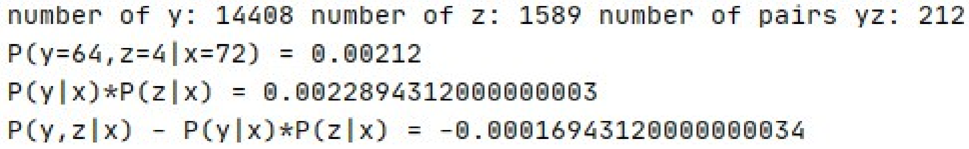
We implemented the uniform error channel using the function legittimate_channel(x) the legitimate channel introduces at most 1 binary error per word, Legitimate a random integer for choosing which is the error and XOR with the the input and BitArray(bin=errors[index]).uint transforms binary string to integer. and eve_channel(x) for the eavesdropper channel introduces at most 3 binary error per word, and the main() function contains the variables x = ”01001000” y = [] z = [] contyz = 0 conty = 0 contz = 0 n = 25000 after that we call the function legittimate_channel(x) and adds new word to a list y.append(word_y) and Same for Eve by calling eve_channel(x) after the all list is appended we Verify the conditional independence and uniformity and print the output and plot the graphic for Legitimate channel and plot the graphic for Eavesdropper channel.

Verify the conditional independence and uniformity of our outputs, P(y,z—x) = P(y—x)*P(z—x), in our experiment we used y = 64 and z = 4 (chosen randomly). We obtained following results:

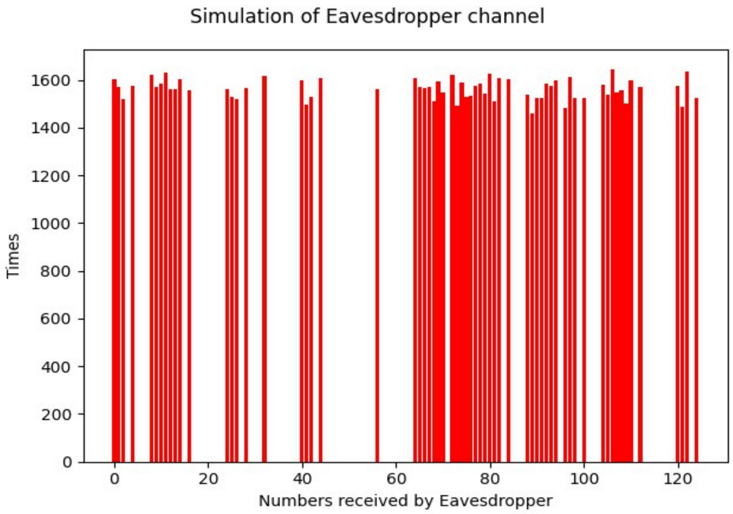


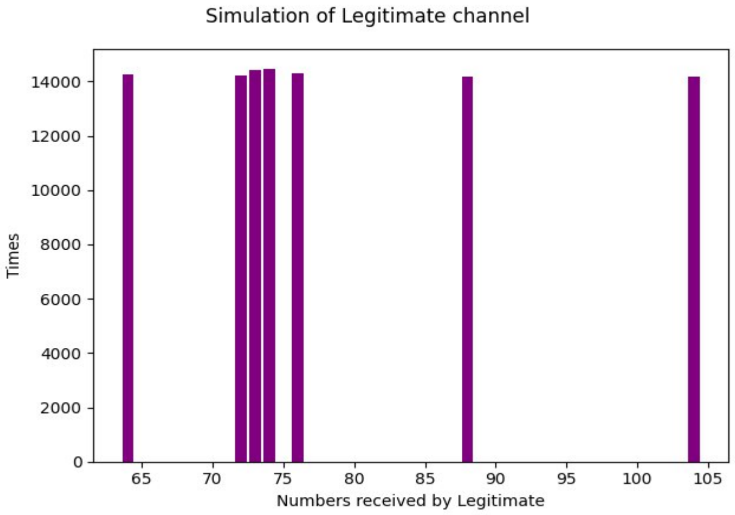
```
number of y: 3611 number of z: 419 number of pairs yz: 48
P(y=64,z=4|x=72) = 0.00192
P(y|x)*P(z|x) = 0.002420814400000004
P(y,z|x) - P(y|x)*P(z|x) = -0.0005008144000000004
```

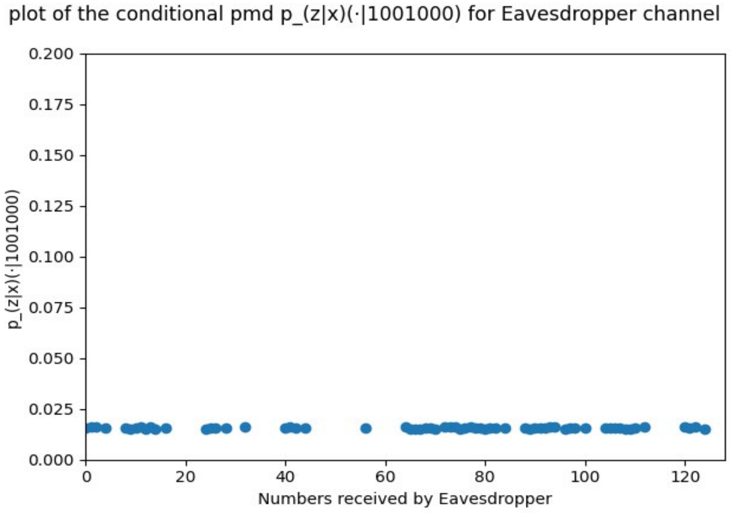
And we iterated our algorithm 25 000 times and we get this output.



```
number of y: 14408 number of z: 1589 number of pairs yz: 212
P(y=64,z=4|x=72) = 0.00212
P(y|x)*P(z|x) = 0.0022894312000000003
P(y,z|x) - P(y|x)*P(z|x) = -0.00016943120000000034
```







Task 2

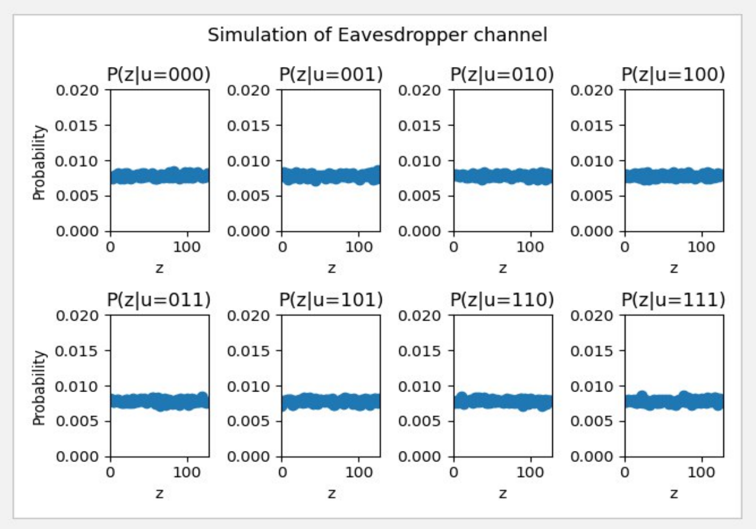
In task 2 we are using the random lib for Generate pseudo-random numbers and BitArray lib for efficient arrays of booleans for better implementation, we defined X array variable for storing the codewords, and `rand_encoder(d)` function for Implement the random binning encoder, first we checks which codeword starts with the prefix and gets it and Calculates the complement of the binary given as input and the codeword x is chosen randomly and uniformly within the bin associated to the message u, to optimise the code we could compute the complement if, and only if, the rand == 1, in this i preferred to show the two choices and how the randomness choice it.

Task 3

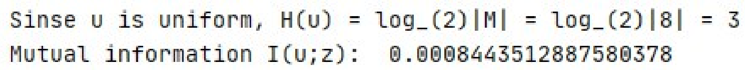
In task we are using the `Numpy` and `BitArray` for array operations G is a 4 linear independent codewords (as columns) first 4 rows form the identity matrix 4x4 and H is the parity check matrix built starting from G last 3 rows of G + Identity 3x3 *c*oset_leader look-up table for choosing the coset leader of the syndrome computed in a paper using H and all of possible choices of 3 bits [x x x], *i*nputs is all of possible inputs, and `rand_decoder(y_string)` is function for implement the legitimate decoder, so that it accepts an input and produces the corresponding output.

Task 4

Here in task 4 the tasks are divided into two parts first part we are using 8X128 matrix to collect the statistics of PDM , pz—u after that we are running simulations by sending the message through the eavesdropper branch of the channel and thereby collecting the distribution pz—u, then we will plot the result with the Number as x and Probability as y axis using z_pmds[i], Here the message(u) is not taken randomly so we are estimating the distribution pz—u.



In the second part ,we are running some simulations by taking a random message and observe the z received by the eavesdropper. Here both u and z are random variables.Then we collect the distribution pu—z .After that we compute the marginal distribution, entropies and the mutual information and we use the help function to compute the following rand_encoder(), eve_channel(). the estimates of H(u), and I(u; z)



```
Sinse u is uniform, H(u) = log_(2)|M| = log_(2)|8| = 3
Mutual information I(u;z):  0.0008443512887580378
```

Considerations 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”)?

Because the legitimate channel introduce at most 1 error and we can always detect it and find the correct input. the eve channel can generate at most 3 errors, he wont understand which bit was ‘flipped’.

2) Is it possible to obtain 4 secret bits per channel use? If so, how should you change your encoder/decoder? If not, why?

No, we cannot have more secrets bits than we have sent as input

3) Is it possible to obtain 2 secret bits per channel use? If so, how should you change your encoderdecoder? If not, why?

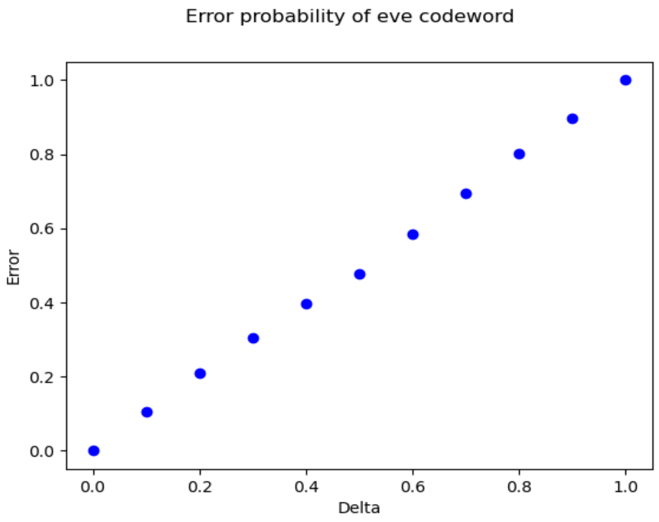
Yes because at most we have 3 secure bits (upper-bound), maybe having a deterministic encoderdecoder

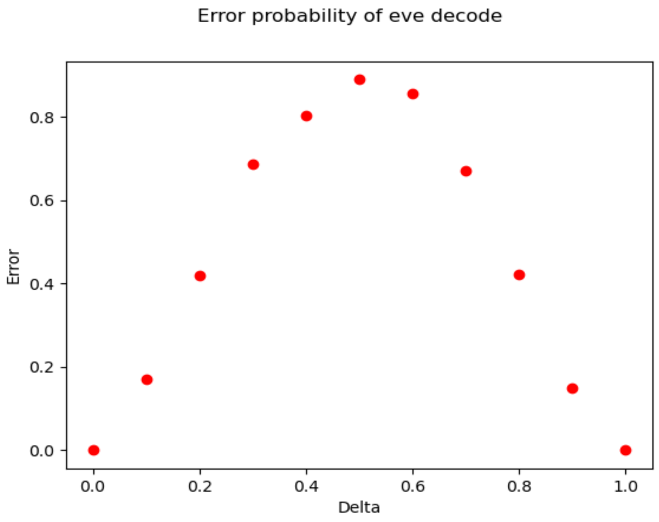
4)One could consider evaluating the secrecy of this mechanism by cascading the eavesdropper channel with a decoder and measuring the resulting error rates. What do you expect Eve’s error would be? Why resort to (more complicated) evaluating the mutual information?

Since Eve cannot know which bits are flipped, we suppose that Eve has to choose randomly among all possibility (1/2$\hat{3}$). So the error is 1-1/2$\hat{3}$

Task 5

Task 5 is a binary symmetric channel that flips bits in bob and eve channels with probability epsilon and delta. The bits to flip are chosen uniformly in the input with a simple random generation of 0 and 1, where 1 means that a flip will occur in that position. This mask is applied with a XOR. The channel is tested first with a long string of bits and the number of flipped bits equals in percentage the values provided for epsilon and delta. Then the full range of inputs for the encoder and decoder of task 2 and 3 is tested multiple times and the results are averaged for better accuracy. The results show that with sufficiently high values of epsilon bob's decoder can fail as more than 1 bit is flipped and the Hamming code can, at most, correct one error. The message decoded by eve depends on the input codeword and delta. The code logs both wrong decoded codewords and the number of flipped bits in the output channels, to provide an insight of the decoding mechanism that relies also on the complementary of the input codewords. Two plots are provided at the end to show these results, by enabling the corresponding if statement.

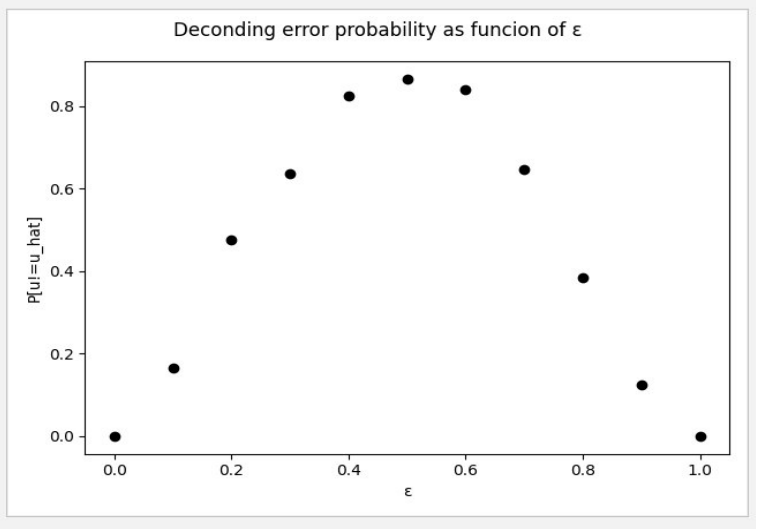




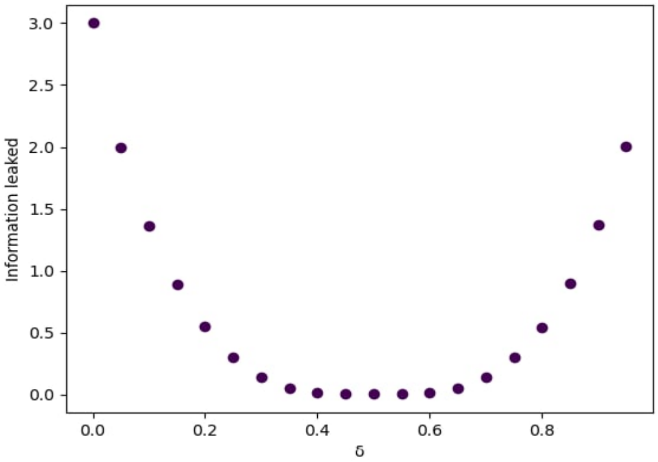
Task 6

In task 6 we use the BSC channel instead of the uniform channel to realize our communications. Similarly to the plot for task5, bob's error probability that it decodes a wrong message ranges from 0 when epsilon reaches 0 and 1, to close to 1 when it approaches 0.5. Note that the error probability does not reach 1 because we need to account for the probability that 0 or only 1 bits are flipped in the whole codeword even when epsilon equals 0.5, and thus it is still decoded correctly. It is instead exactly 0 with epsilon equals to 0 or 1 because in this case we are sure no bits (or all in the case epsilon equals 1) are flipped and computations are repeated to increase accuracy also here.

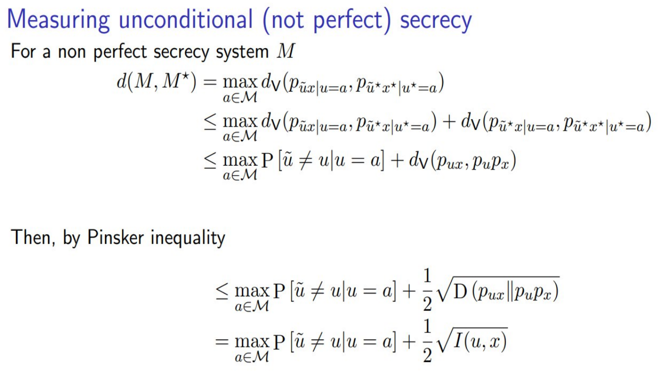
A plot of the error decoding probability P as a function of ε for the BSC



A plot of the mutual information I as a function of ε for the BSC



For point 1.3.6 we need this formula:



Measuring unconditional (not perfect) secrecy

For a non perfect secrecy system M

$$\begin{aligned} d(M, M^\star) &= \max_{a \in \mathcal{M}} d_{\mathsf{V}}(p_{\tilde{u}x|u=a}, p_{\tilde{u}^\star x^\star|u^\star=a}) \\ &\leq \max_{a \in \mathcal{M}} d_{\mathsf{V}}(p_{\tilde{u}x|u=a}, p_{\tilde{u}^\star x|u^\star=a}) + d_{\mathsf{V}}(p_{\tilde{u}^\star x|u=a}, p_{\tilde{u}^\star x^\star|u^\star=a}) \\ &\leq \max_{a \in \mathcal{M}} \mathrm{P}\left[\tilde{u} \neq u | u = a\right] + d_{\mathsf{V}}(p_{ux}, p_u p_x) \end{aligned}$$

Then, by Pinsker inequality

$$\begin{aligned} &\leq \max_{a \in \mathcal{M}} \mathrm{P}\left[\tilde{u} \neq u | u = a\right] + \frac{1}{2}\sqrt{\mathrm{D}\left(p_{ux} \| p_u p_x\right)} \\ &= \max_{a \in \mathcal{M}} \mathrm{P}\left[\tilde{u} \neq u | u = a\right] + \frac{1}{2}\sqrt{I(u, x)} \end{aligned}$$