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 8 Python source files: `main.py` contains all the function for tasks, `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

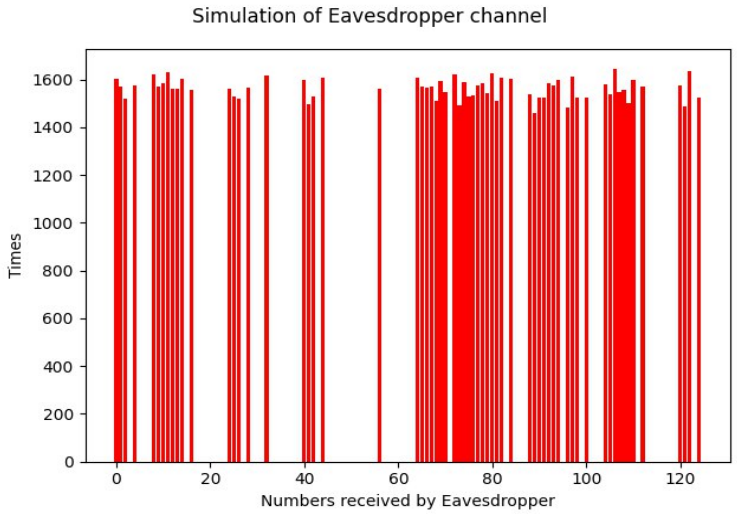
We implemented the uniform error channel using the function `legittimate_channel(x)` the legitimate channel introduces at most 1 binary error per word, Legitimate a random integer for choosing which is the error and XOR with the the input and BitArray(bin=errors[index]).uint transforms binary string 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.

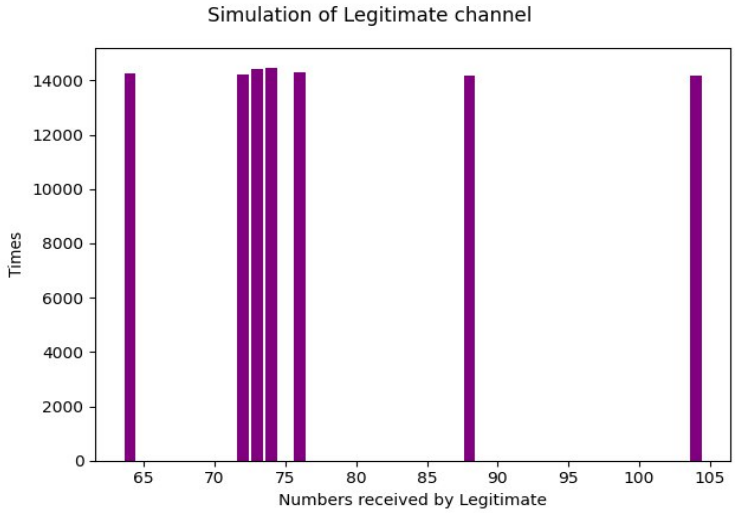
Verify the conditional independence and uniformity of our outputs, P(y,z—x) = P(y—x)*P(z—x), in our experiment we used y = 64 and z = 4 (chosen randomly). We obtained following results:

```
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) = 0.0024208144000000004
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

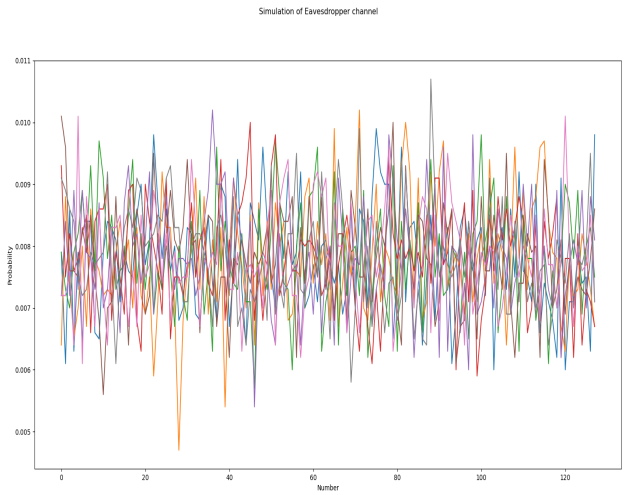
In task 2 we are using the random lib for Generate pseudo-random numbers and BitArray lib for efficient arrays of booleans for better implementation, we defined X array variable for storing the codewords, and `rand_encoder(d)` function for Implement the random binning encoder, first we checks which codeword 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().

Task 5

In task 5 we are using NumPy, bitstring, utils for string to array and array to string, math, matplotlib for plot and statistics libraries for better implementation and we import the task2 and task3 to connected in between the random binning encoder developed in Task 2, and the decoder developed in Task 3, for test with task 2/3 encoder decoder we defined deltas as np.arange(0, 1.1, 0.1), epsilon as 0.1, wrong_eves to store number of wrong codewords decoded by eve in percentage, wrong_bobs to store number of wrong codewords decoded by bob in percentage, codeword_error_eve for % of

wrong bits in the codeword received by eve after we compute the channel capacity then we defined multi_error_codeword_count to count how many pairs of codewords (x, y)have more than 1 bit different, error_bob_count for wrong codewords decoded by bob and this is not the same as multi_error_codeword_count, because the complementary (+- 1 bit) of the codewords are decoded correctly, error_eve_count for wrong codewords decoded by eve and codeword_error_eve_cycle to count how many flipped bits are in the codeword received by eve at each cycle after that repeat multiple times for better estimates then compare decode with input.

Task 6