Padua (PD)
28th April 2025

INFORMATION SECURITY LAB.2
A.Y 2024/2025

Università degli Studi di Padova

Lab 2 - Information Security

Cyberducks
Andrea Andreozzi (2163406)
Riccardo Scalco (2155352)
Sergio Cibecchini (2155353)
Luca Ferrari (2166294)

1 Task 1 - Uniform Error Wiretap Channel

In this task we were asked to simulate a wiretap channel. Given the input and output alphabets being 7-bit words with $\mathcal{X} = \mathcal{Y} = \mathcal{Z} = \{0,1\}^7$ the wiretap channel should be implemented s.t the legitimate channel introduced *at most 1 binary errors* and the eavesdropper channel would instead face at most *3 binary errors* . The channel would also have to behave in such a way that, when tested with a sufficiently large number of transmissions ($n \geq 10^4$), the channel would produce **uniform and independent outputs**. To achieve this we implemented the following functions:

- `generateAllErrors(num_bits, max_errors)` to generate all possible channel errors
- `getRandomElement(vector)` to get a random error from the array of random errors
- `xor_between_vectors(a, b)` to apply the channel error to the input binary word

We repeated this procedure for `n_iterations = 100000` and plotted the results as shown in Figures 1 and 2

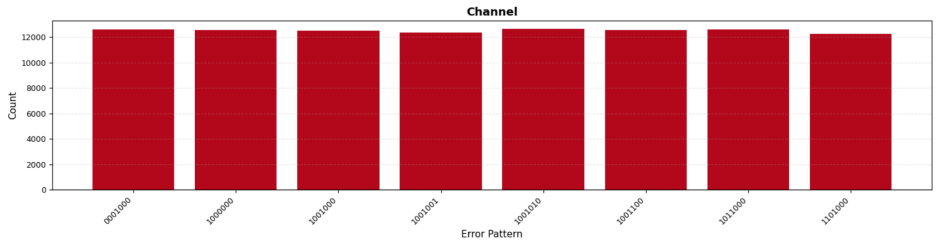

Figure 1: Distribution of outcomes for the legitimate channel

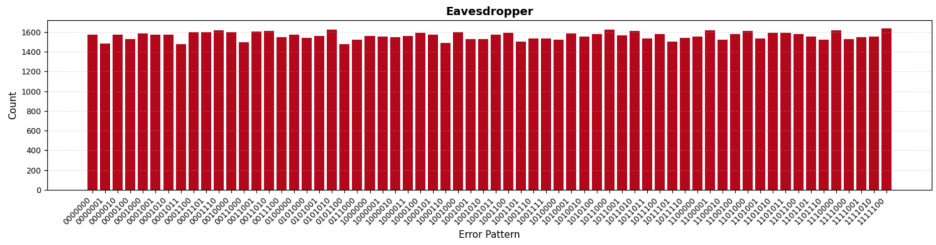

Figure 2: Distribution of outcomes for the eavesdropper channel

Thanks to these plots, we were able to empirically verify the uniformity of the outputs of our simulated channel, both for the legitimate user and the eavesdropper.

2 Task 2 - Random Binning Encoder

In this task we were asked to implement a Random Binning Encoder. We were given a binary message $u \in \mathcal{M} = \{0,1\}^3$, and a hamming code $\mathcal{X}''$ where the minimum *Hamming distance* is 3. As instructed, the function, `def generate_codewords(u)` applied the following operations to the message:

- Appends a zero to the front of the binary message
- Looks for the matching $4-bit$ prefix among the members of the Hamming Code
- Finds the appropriate member, computes the binary complement by applying a `xor` with a $7-bit$ string of all ones
- Returns one at random between the binary string and its complement

For example, with the word $u = 100$ we get, as expected:

$$T_{x|u} = \{1011010, 0100101\}$$

3 Task 3 - Random Binning Decoder

In this task we were asked to create the **decoder** for the Random Binning Encoder that we implemented in Task 2. The goal of this decoder was to reconstruct the original message from the input `7-bit` binary string passed trough the channel and potentially correct `1-bit` errors. Our implementation proceeded as following:

- `def minimum_distance_with_code(x, all_word_code)` compares the input message `x` with all the possible words of the Hamming Code, finds the one with the smallest distance and returns it
- `def find_real_input(codeword)` checks the first bit to see which of the two versions of the word we have, removes the first leading and the last three trailing bits, applies the `xor` if necessary and returns the transmitted message

As expected, both with input 1011010 and 0100101 we get 100.

4 Task 4 - Verify Perfect Secrecy

In this task we had to prove empirically that the output z of the eavesdropper channel is independent of the secret message u, i.e. show that for every possible message each output received by the eavesdropper was equally likely.
Firstly we generated all possible *3-bit* messages and ended up with $2^3 = 8$ messages. Then, for each message, we proceeded in this way:

a. Encode the message trough the `encoder(u)` function

b. Pass the message trough the eavesdropper channel

c. Log the output and increase its counter

This pipeline was repeated for $N = 1000 \cdot |\mathcal{Z}| \simeq 10^5$ realizations. The output below shows that the various possible outputs relative to a specific input are all (almost) equally likely for all possible values of d.

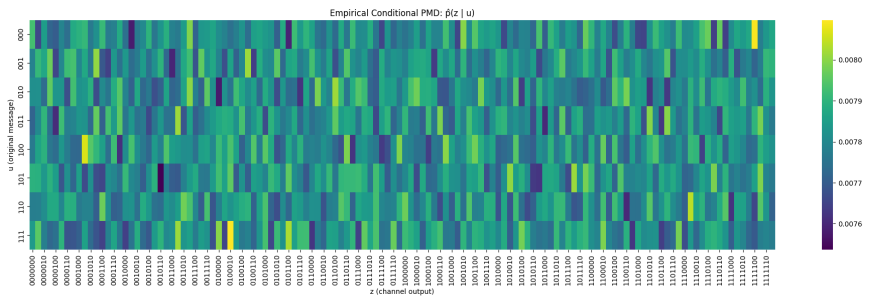

Figure 3: Distribution of outcomes for all possible messages trough the eavesdropper's channel

Next, we needed to compute the distributions empirically. Firstly, we computed the joint distribution $\hat{p}_{u,z}(d,c)$ by counting the number of times each (u,z) couple appeared and dividing by the total number. Then we extracted the marginal distribution over u through the formula:

$$\hat{p}_u(d) = \sum_c \hat{p}_{u,z}(d,c)$$

And the same for z. Through this information, we were able to calculate $H(u) = E[i_u(u)] = 3$ and the mutual information:

$$\hat{I}(u,z) = I(u;z) = \sum_{d\in\mathcal{M},c\in\mathcal{Z}} \hat{p}_{u,z}(d,c)\log_2\left(\frac{\hat{p}_{u,z}(d,c)}{\hat{p}_u(d)\hat{p}_z(c)}\right) = 0.000828 \simeq 0$$

Which confirms our hypothesis of statistical independence of u and z.
With the current scheme, we transmit 3 secret bits per channel use (since each encoded word corresponds to a 3-bit message) and approximately $3/7 \approx 0.4286$ secret bits per transmitted bit (as each codeword has 7 bits). It is **not** possible to achieve 4 secret bits per channel use with the current 7-bit code, because the encoder can at most map $2^3 = 8$ messages reliably given the code structure and decoding constraints. Achieving 2 secret bits per channel use would be possible by modifying the encoder to map 2 bits (instead of 3) to codewords. If we were to cascade the eavesdropper's output with a decoder, we'd expect Eve's error probability to be high because of the nature of the eavesdropper channel.

5 Task 5 - Binary Symmetric Channel

In this task, we had to switch our communication channel from the uniform error channel that we previously implemented to a *binary symmetric channel*. This means we had to use the same encoding and decoding methods, but the transmission now had a probabilistic error rate based on two parameters, ϵ and δ. In our case, we arbitrarily chose $\epsilon = 0.1$ and $\delta = 0.3$.
We implemented a `bsc(input, probability_error)` function, which flips each bit with the given probability of error. We then tried the `bsc` channel on an input string of length $N = 50000$ and received respectively:

```
Number of errors in legitimate channel (with epsilon=0.1): 4998. BER = 0.09996
Number of errors in eavesdropper channel (with delta=0.3): 14941. BER = 0.29882
```

Moreover, we test the `bsc` channel with 500 different inputs of size 3 and obtain the following result:

```
Number of errors in legitimate channel (with epsilon=0.1): 71
Number of errors in eavesdropper channel (with delta=0.3): 330
```

Since the error rate is no longer deterministically under a quantity that the Hamming code is able to correct (`num_errors <=1`), we no longer have perfect reliability and secrecy as the error distribution is no longer uniform.

6 Task 6 - Security Evaluation Over BSC

In this task we had to evaluate the security of the just implemented BSC. First we defined a series of ϵ and δ pairs. The closer the two values, the less advantage the legitimate channel has over the eavesdropper. These are the values we chose:

$$[(0.01, 0.55), (0.05, 0.50), (0.1, 0.45), (0.15, 0.40), (0.20, 0.35), (0.25, 0.3), (0.3, 0.3)]$$

Then we simulated, for each ϵ a sequence of transmissions and compared the sent message with the message that was decoded and computed the error rate. The following graph shows the probability of the original message u being different from the recovered message $\hat{u}$ at a variation of ϵ:

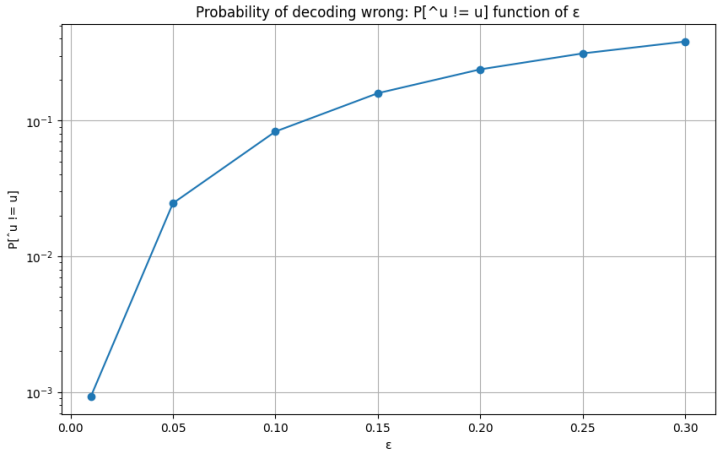

Figure 4: Probability of error in decoding the message with increasing ϵ for the legitimate channel

In this case, since the number of errors is no longer deterministic, we had no longer guarantees in therms of reliability and secrecy. We then calculated the empirical distribution with the same method used in Task 4 but using the BSC channel instead of the uniform error channel. For each case, we computed the mutual information and entropy. The entropy, as expected, did not change for the various iterations, remaining $H(u) = 3$.
Also as expected, the mutual information decreases as the eavesdropper channel quality deteriorates:

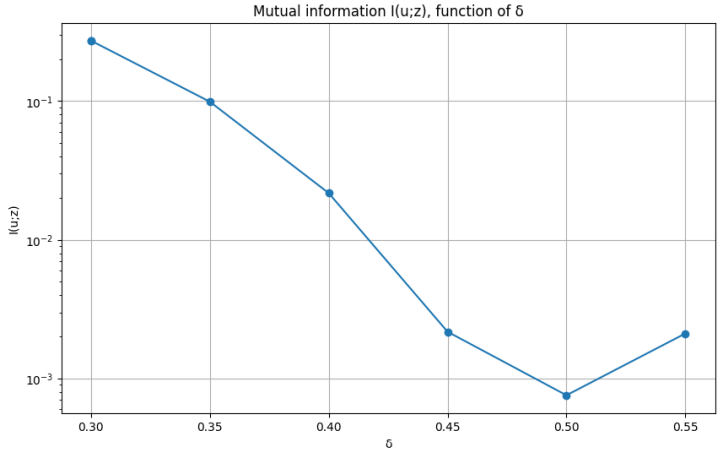

Figure 5: Mutual Information at a variation of the delta for the eavesdropper

Finally, we analyzed the distinguishability between the system and the ideal counterpart by computing the *Total Variational Distance* in function of ϵ and δ.

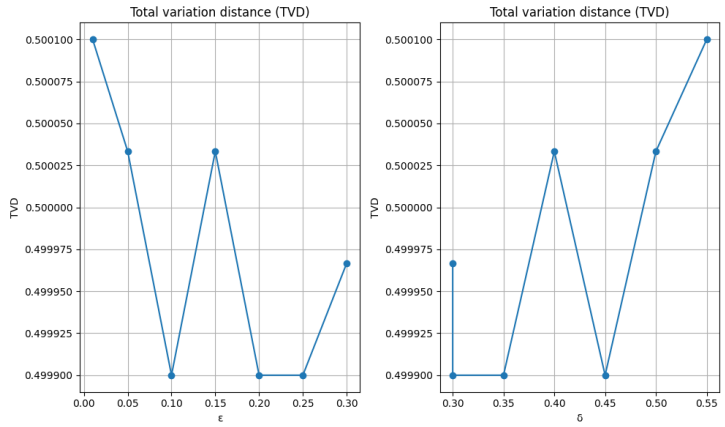

Figure 6: Total variational distance in function of delta and epsilon

And finally, we assessed the security level for our security mechanisms trough the same graph analyzed in class:

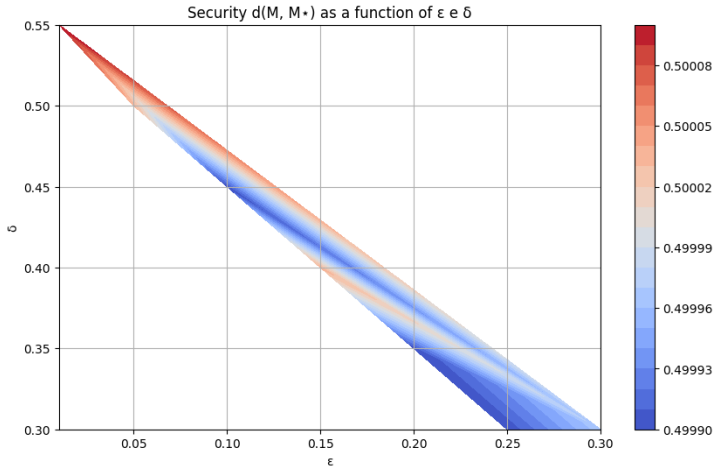

Figure 7: Security level

7 Task 7 - Simulate transmission over a wiretap AWGN channel

In this task, we had to implement a PAM modulator/demodulator and simulate an AWGN channel.
So we developed the following methods:

- `pam_modulate()`
- `awgn_channel()`
- `pam_demodulate()`

Subsequently, we implement the complete chain:
`encoder > pam modulator > agwn channel > pam demodulator > decoder`
Then we test the complete chain with a value `SNR=10db` and an input length of *300 bits*. We obtain a bit error rate `BER=0.365`.
Finally we test our script with different values of `SNR` and for all values we calculate the `BER`. We plot the result and we obtain this graph:

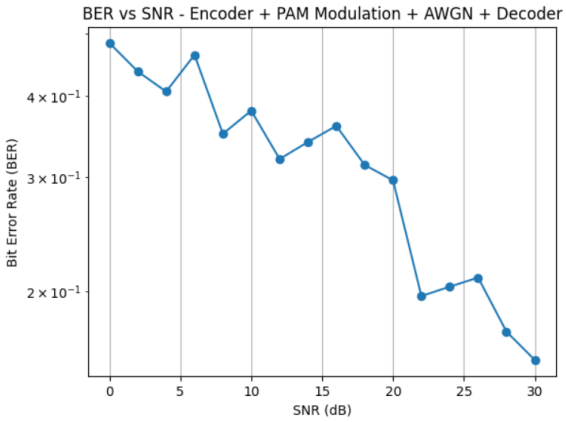

Figure 8: BER with different values of SNR

As expected, even in this case, perfect secrecy and reliability are not guaranteed, as the `BER` heavily depends on the value of the `SNR`

8 Task 8 - Evaluate the system security over the wiretap AWGN channel

For developing this task, we define two cases:

- We define a fix value of Eve SNR (0 db) and a variabile SNR for Bob (from 0db to 20db). So the eavesdropper channel is worse than the legitimate channel
- In the second case we did the opposite: we fix the BOB SNR to 20 db and we try we different values of Eve SNR (again from 0db to 20 db)

We obtain the following plot:

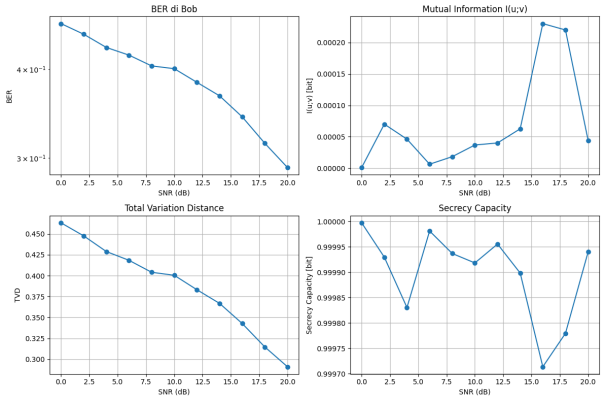

Figure 9: Fix Eve SNR = 0db. Variable Bob SNR

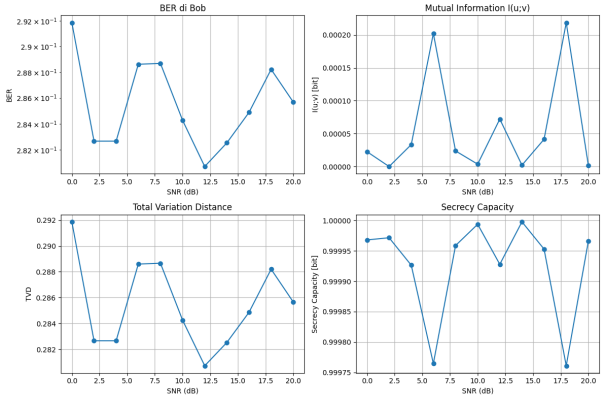

Figure 10: Fix Bob SNR = 20db. Variable Eve SNR