Lab2 RoC Information Security Report

Group: TCP

Cavasin Saverio - ID. 2044731 Cosuti Luca - ID. 2057061
De Faveri Francesco Luigi - ID. 2057069

A brief description of the report

All the tasks were developed in Python and verified to assess correctness. For each task, a `.py` file was written, while the answers to the "Considerations and remarks" are provided in the Section on page 5. Based on the specific requirements for each one of them, the provided implementations both defined the structuring methods and the related empirical proofs. Different kinds of plots and visual representations are introduced as well, in order to summarize and show the statistical relationships between the elements that characterize these frameworks and models.

Task 1: Wiretap Channel

We modeled a wiretap channel by defining a function based on the xor bitwise operation. The sides of such computation are the required input and a randomly generated error, related to either the legitimate or eavesdropped channel. The former introduces at most 1 binary error per word, and the second at most 3.

Formally:

$$\textbf{input} \bigoplus \textbf{channel random error}$$

with:

$$\textbf{input} = \left[1, 0, 0, 1, 0, 0, 0\right]$$

$$\textbf{Bob\_channel\_errors} = \left\{\begin{matrix} [0,0,0,0,0,0,0], & [0,0,0,0,0,0,1], & [0,0,0,0,0,1,0], \\ [0,0,0,0,1,0,0], & [0,0,0,1,0,0,0], & [0,0,1,0,0,0,0], \\ [0,1,0,0,0,0,0], & [1,0,0,0,0,0,0] & \end{matrix}\right\}$$

$$\textbf{Eve\_channel\_error} = \left\{\begin{matrix} [0,0,0,0,0,0,0], & [0,0,0,0,0,0,1], & [0,0,0,0,0,1,0], \\ [0,0,0,0,1,0,0], & [0,0,0,1,0,0,0], & [0,0,1,0,0,0,0], \\ [0,1,0,0,0,0,0], & [1,0,0,0,0,0,0], & [0,0,0,0,0,1,1], \\ [0,0,0,0,1,0,1], & [0,0,0,1,0,0,1], & [0,0,1,0,0,0,1], \\ [0,1,0,0,0,0,1], & [1,0,0,0,0,0,1], & [0,0,0,0,1,1,0], \\ [0,0,0,1,0,1,0], & [0,0,1,0,0,1,0], & [0,1,0,0,0,1,0], \\ [1,0,0,0,0,1,0], & [0,0,0,1,1,0,0], & [0,0,1,0,1,0,0], \\ [0,1,0,0,1,0,0], & [1,0,0,0,1,0,0], & [0,0,1,1,0,0,0], \\ [0,1,0,1,0,0,0], & [1,0,0,1,0,0,0], & [0,1,1,0,0,0,0], \\ [1,0,1,0,0,0,0], & [1,1,0,0,0,0,0], & [0,0,0,0,1,1,1], \\ [0,0,0,1,0,1,1], & [0,0,1,0,0,1,1], & [0,1,0,0,0,1,1], \\ [1,0,0,0,0,1,1], & [0,0,1,1,0,0,1], & [0,1,0,1,0,0,1], \\ [1,0,0,1,0,0,1], & [0,1,1,0,0,0,1], & [1,0,1,0,0,0,1], \\ [1,1,0,0,0,0,1], & [0,1,1,1,0,0,0], & [1,0,1,1,0,0,0], \\ [1,1,0,1,0,0,0], & [1,1,1,0,0,0,0] & \end{matrix}\right\}$$

After iterating such a process over a dictionary of no less than 10000 elements, we extracted and presented the mean and standard deviation of the channels while injecting the suggested input. We can see that the number of occurrences of each output is consistent and uniformly distributed. Indeed, the use of pythons' **random.choice()** method allows to uniformly pick one error each time.

Mean number of occurrences per encoded message in the legitimate channel: 1250

Mean number of occurrences per encoded message in the eavesdropper channel: 227.27

Standard deviation of the legitimate channel: 26.11

Standard deviation of the eavesdropper channel: 16.52

Task 2-3: Random binning encoding/decoding

Random binning was implemented starting from the definition of the proper cipher and message spaces.

$$\textbf{cipher\_space} = \left\{\begin{matrix} [0,0,0,0,0,0,0], & [1,0,0,0,1,1,0], & [0,1,0,0,1,0,1], \\ [0,0,1,0,0,1,1], & [0,0,0,1,1,1,1], & [1,1,0,0,0,1,1], \\ [1,0,1,0,1,0,1], & [1,0,0,1,0,0,1], & [0,1,1,0,1,1,0], \\ [0,1,0,1,0,1,0], & [0,0,1,1,1,0,0], & [1,1,1,0,0,0,0], \\ [1,1,0,1,1,0,0], & [1,0,1,1,0,1,0], & [0,1,1,1,0,0,1], \\ & [1,1,1,1,1,1,1] & \end{matrix}\right\}$$

$$\textbf{message\_space} = \left\{\begin{matrix} [0,0,0], & [0,0,1], & [0,1,0], & [0,1,1], \\ [1,0,0], & [1,0,1], & [1,1,0], & [1,1,1] \end{matrix}\right\}$$

The encoding of a message is built upon the division, manipulation, and concatenation of two 4-bit words as requested. Eventually, the model is assessed for correctness by checking the membership of the computation in the cipher space.

Decoding was provided in a similar fashion. Additionally, we now cascaded the coder and decoder both standalone and jointly with the legitimate channel. This way we were able to confirm the Hamming code being systematic and correct by assessing the presence of the respective encoded/decoded components in the cipher/message spaces.

Task 4: Perfect Secrecy

Starting from the eavesdropper error corruption we defined in task1, we built a vector of "corrupted messages", i.e. as many corrupted transformations of the cipher_space elements as we please (e.g. 15k). This way, we are able to model the secrecy mechanism of the required framework when it comes to what information the eavesdropper can gather. Furthermore, to analyze which kind of probability distribution this information provides, we create and compare counters of such looped instances. And we show the frequencies of the relative transformations given each cipher-text.

Again, as shown by the plots we empirically proved that the eavesdropper output are independent from whatever plaintext was fed into the system.

Moreover, the modelization trought the **random.choice()** method ensures a uniform distribution for both the plaintexts and the corrupted words.

Hence, we can approximate the marginal dstributions $\hat{p}_u(d) = \frac{1}{|\mathcal{M}|} = \frac{1}{7}$ and $\hat{p}_z(c) = \frac{1}{|\mathcal{Z}|} = \frac{1}{128}$. By the argued independence, we therefore then set

$$\begin{aligned} \hat{p}_{u,z}(d,c) &= \hat{p}_u(d) \cdot \hat{p}_z(c) \\ &= \frac{1}{7} \cdot \frac{1}{128} \\ &\approx 0,0011 \end{aligned}$$

Considering how we discussed the empirical independence of our random variables, we can deduct that the empirical mutual information $\hat{I}(u,z)$ can be computed as:

$$\begin{aligned} \hat{I}(u,z) &= \sum_{d\in\mathcal{M}\ ,\ c\in\mathcal{Z}} \hat{p}_{u,z}(d,c) \cdot \log_2 \Big(\frac{\hat{p}_{u,z}(d,c)}{\hat{p}_u(d)\ \hat{p}_z(c)}\Big) \\ (independence) &= \sum_{d\in\mathcal{M}\ ,\ c\in\mathcal{Z}} \hat{p}_{u,z}(d,c) \cdot \log_2 \Big(\frac{\hat{p}_u(d)\ \hat{p}_z(c)}{\hat{p}_u(d)\ \hat{p}_z(c)}\Big) \\ &= \sum_{d\in\mathcal{M}\ ,\ c\in\mathcal{Z}} \hat{p}_{u,z}(d,c) \cdot \log_2(1) \\ &= 0 \end{aligned}$$

and the empirical entropy $\hat{H}(u)$ can be computed as

$$\begin{aligned} \hat{H}(u) &= -\sum_{d\in\mathcal{M}} \hat{p}_u(d) \cdot \log_2 \hat{p}_u(d) \\ &= -\sum_{1}^{7} \frac{1}{7} \cdot \log_2(\frac{1}{7}) \\ &\approx -\sum_{1}^{7}\ \frac{1}{7} \cdot -2.807 \\ &=\ 7 \cdot\ \frac{1}{7} \cdot 2.807 \\ &\approx\ 2.81 \end{aligned}$$

Task 5: Transmission over a Binary Symmetric Channel (BSC)

We implemented the Binary Symmetric Channel through the definition of arbitrary ε, δ values. From such values we implemented a function to apply corruption based on the relationships between such values, through the absolute differences w.r.t. $\frac{1}{2}$. Such a framework allowed us to scale and apply the degradation on either the legitimate or eavesdropper channel. See theorem Shannon.
In `Task5.py`, we manually choose a fixed δ and ε to be the "bias" of the BSC and we computed the probabilities of **correctly decoded** and **mismatched** as functions of the biases.

Task 6: Evaluation of the system security over the wiretap BSC

Implementing the code developed for tasks 3 and 4, jointly with the definitions of both the parameters and the corruption functions of task 5, we were able to provide accordingly the analysis of the reliability and secrecy concerning, respectively, the legitimate channel's transmitted bits and the possible eavesdropped information.

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")?**

A. We proved that our observed outputs are empirically independent w.r.t. the plain-texts, as shown by our plots. Therefore, in order to answer this question we propose the following strategy. Out of simplicity, we fix a plaintext to get to the channel use ("transmitted word") we want to analyse (as we said, we prove the results are empirically uniform throughout all our message space, therefore the analysis we propose will hold no matter the choice in the same fashion), e.g. 000. We then compute the number of occurrences in specific events that allow us to discern interesting proprieties. I.e. in this case, we know that given the proposed implementation in the cryptosystem and in the Hamming code, 000 can only be encoded as 0000000 or 1111111. This means that, by observing the realizations in the outputs for this plaintext, we can immediately infer which of the two cases we are dealing with as the transmitted word, moreover we are then able to recall how many bits were changed / how many secret bits were injected through the channel.

Remark: Considering the nature and definition of the Hamming space (7,4), we notice that each of the elements in it has a Hamming distance less or equal to 3 w.r.t. either 0000000 or 1111111. Additionally, we can see that every possible binary combination of 7 bits can be achieved starting from one of those two words, with a Hamming distance $\in \{0, 1, 2, 3\}$.

1. 0000000 has Hamming distance 0 from 0000000
2. one 1 digit and six 0 digits : Hamming distance 1 from 0000000
3. two 1 digit and five 0 digits : Hamming distance 2 from 0000000
4. three 1 digit and four 0 digits : Hamming distance 3 from 0000000
5. four 1 digit and three 0 digits : Hamming distance 3 from 1111111
6. five 1 digit and two 0 digits : Hamming distance 2 from 1111111
7. six 1 digit and one 0 digits : Hamming distance 1 from 1111111
8. 1111111 has Hamming distance 0 from 1111111

Hence, to tell how many secret bits per channel use we obtained through the eavesdropper output we count the instances in which we observed, w.r.t. one of the two hamming-code allowed encodings :

1. 0 bit changed
2. 1 bit changed
3. 2 bit changed
4. 3 bit changed

For instance, let's consider the output x = [0,0,0,0,1,0,1]. We showed that it occurred 402 times during our 10k run of the algorithm. Considering that the maximum amount of bits that are allowable to change / be injected in our scheme is 3 we can immediately say the related encoding in the given Hamming-code was 0000000 and the fifth and seventh bits were changes. Hence we will say that here we have **two** secret bits.

Now we generalize the same process to all the observed eavesdropper outputs and we propose a summary table, (the results were randomly generated and stored a test run), from which it is both possible to extract additional statistics and confirm the independent and uniform distribution of the eavesdropped outputs (w.r.t. the messages).

Observation : Count : Hamming Distance

$$\mathbf{000} = \left\{ \begin{array}{lll}
(0,1,0,0,0,0,0) : 125 : 1 & (1,1,1,0,0,0,0) : 116 : 3 & (1,0,0,0,1,1,0) : 100 : 3 \\
(1,1,0,1,1,0,1) : 107 : 2 & (1,0,0,0,0,0,0) : 122 : 1 & (1,1,0,1,1,1,0) : 114 : 2 \\
(0,1,1,1,1,1,1) : 134 : 1 & (0,1,1,0,0,0,0) : 125 : 2 & (1,1,1,1,0,1,1) : 126 : 1 \\
(1,1,0,0,1,1,1) : 101 : 2 & (1,1,0,0,1,1,0) : 112 : 3 & (0,1,0,0,1,1,0) : 121 : 3 \\
(1,0,0,1,1,1,0) : 113 : 3 & (1,1,1,0,1,0,0) : 125 : 3 & (0,0,0,1,1,1,1) : 111 : 3 \\
(0,0,1,1,1,1,0) : 106 : 3 & (0,1,1,0,1,0,1) : 110 : 3 & (1,1,1,1,1,1,1) : 100 : 0 \\
(0,1,1,1,1,0,0) : 120 : 3 & (0,1,1,0,1,0,0) : 104 : 3 & (1,0,0,1,1,1,1) : 118 : 2 \\
(0,1,0,1,0,1,0) : 125 : 3 & (0,1,1,1,0,0,1) : 118 : 3 & (1,0,1,1,1,0,1) : 106 : 2 \\
(1,0,1,1,1,1,0) : 126 : 2 & (1,0,1,1,1,0,0) : 129 : 3 & (0,1,1,1,0,1,0) : 122 : 3 \\
(1,0,1,1,0,1,1) : 125 : 2 & (0,0,1,1,1,0,0) : 107 : 3 & (1,1,0,0,0,1,0) : 116 : 3 \\
(1,0,1,0,0,1,0) : 115 : 3 & (0,0,0,0,1,0,1) : 128 : 2 & (1,1,1,0,0,1,0) : 109 : 3 \\
(1,1,0,1,0,1,1) : 120 : 2 & (0,0,0,1,1,1,0) : 109 : 3 & (0,1,0,1,1,0,1) : 103 : 3 \\
(1,0,1,0,0,0,1) : 146 : 3 & (1,1,1,1,1,0,0) : 116 : 2 & (0,0,0,1,0,1,1) : 110 : 3 \\
(1,1,1,0,1,0,1) : 98 : 2 & (1,1,0,1,1,1,1) : 125 : 1 & (0,0,1,0,1,1,0) : 124 : 3 \\
(1,0,1,1,1,1,1) : 103 : 1 & (1,1,1,1,0,0,0) : 102 : 3 & (0,1,0,0,1,0,0) : 111 : 2 \\
(0,0,1,1,0,0,0) : 129 : 2 & (1,1,0,1,0,0,0) : 133 : 3 & (0,1,0,1,0,0,1) : 126 : 3 \\
(1,0,0,0,1,0,1) : 124 : 3 & (1,0,1,0,1,0,1) : 107 : 3 & (0,0,1,1,0,0,1) : 112 : 3 \\
(1,0,0,1,0,0,1) : 116 : 3 & (0,1,1,0,0,1,0) : 130 : 3 & (0,0,1,0,0,0,1) : 117 : 2 \\
(0,1,0,0,0,1,0) : 133 : 2 & (1,1,0,0,0,0,1) : 126 : 3 & (0,0,1,0,0,0,0) : 114 : 1 \\
(0,0,1,1,1,1,1) : 122 : 2 & (1,1,1,1,0,0,1) : 95 : 2 & (1,1,1,1,0,1,0) : 125 : 2 \\
(0,1,1,1,0,1,1) : 120 : 2 & (0,1,0,0,0,1,1) : 134 : 3 & (0,0,0,0,1,1,1) : 110 : 3 \\
(0,1,0,0,1,0,1) : 110 : 3 & (1,0,1,0,0,0,0) : 127 : 2 & (1,0,0,1,1,0,1) : 132 : 3 \\
(1,1,0,0,1,0,1) : 110 : 3 & (0,0,0,0,0,0,1) : 123 : 1 & (0,1,1,0,1,1,0) : 128 : 3 \\
(1,1,1,1,1,1,0) : 109 : 1 & (1,0,0,0,1,1,1) : 126 : 3 & (1,0,1,0,1,1,1) : 81 : 2 \\
(0,1,1,0,1,1,1) : 115 : 2 & (0,1,0,0,0,0,1) : 110 : 2 & (0,0,0,0,0,1,1) : 117 : 2 \\
(1,1,0,0,0,1,1) : 123 : 3 & (1,0,0,1,0,1,0) : 111 : 3 & (1,1,0,0,0,0,0) : 130 : 2 \\
(0,1,0,0,1,1,1) : 107 : 3 & (0,0,0,1,1,0,1) : 123 : 3 & (0,0,1,0,1,0,0) : 133 : 2 \\
(0,1,0,1,1,1,1) : 138 : 2 & (0,0,1,1,0,1,0) : 119 : 3 & (0,0,1,0,0,1,0) : 112 : 2 \\
(0,1,1,0,0,1,1) : 123 : 3 & (1,1,1,1,1,0,1) : 116 : 1 & (1,0,1,0,1,0,0) : 102 : 3 \\
(0,1,1,1,1,1,0) : 112 : 2 & (0,0,0,1,0,0,0) : 129 : 1 & (0,0,0,1,1,0,0) : 117 : 2 \\
(1,1,0,1,0,1,0) : 115 : 3 & (1,1,0,0,1,0,0) : 119 : 3 & (0,1,1,1,0,0,0) : 108 : 3 \\
(0,0,0,0,0,0,0) : 123 : 0 & (0,1,0,1,1,0,0) : 116 : 3 & (1,1,1,0,0,1,1) : 111 : 2 \\
(1,1,0,1,0,0,1) : 120 : 3 & (0,0,1,1,0,1,1) : 102 : 3 & (1,1,1,0,0,0,1) : 111 : 3 \\
(0,0,0,0,0,1,0) : 105 : 1 & (1,0,1,1,0,0,1) : 120 : 3 & (1,0,1,0,1,1,0) : 122 : 3 \\
(1,1,1,0,1,1,1) : 126 : 1 & (0,0,0,1,0,0,1) : 115 : 2 & (0,0,0,0,1,1,0) : 101 : 2 \\
(1,0,0,0,1,0,0) : 105 : 2 & (0,1,0,1,0,0,0) : 126 : 2 & (0,1,1,0,0,0,1) : 123 : 3 \\
(1,1,1,0,1,1,0) : 117 : 2 & (1,0,0,0,0,0,1) : 106 : 2 & (0,1,1,1,1,0,1) : 134 : 2 \\
(0,0,1,0,1,1,1) : 123 : 3 & (1,0,0,1,1,0,0) : 111 : 3 & (0,1,0,1,1,1,0) : 100 : 2 \\
(0,1,0,1,0,1,1) : 113 : 3 & (1,0,0,1,0,0,0) : 108 : 2 & (1,0,0,1,0,1,1) : 121 : 3 \\
(0,0,0,0,1,0,0) : 106 : 1 & (0,0,1,0,0,1,1) : 92 : 3 & (1,0,1,0,0,1,1) : 121 : 3 \\
(0,0,0,1,0,1,0) : 113 : 2 & (1,0,0,0,0,1,0) : 109 : 2 & (0,0,1,1,1,0,1) : 105 : 3 \\
(1,0,1,1,0,1,0) : 103 : 3 & (1,0,1,1,0,0,0) : 121 : 3 & (1,1,0,1,1,0,0) : 111 : 3 \\
(0,0,1,0,1,0,1) : 117 : 3 & (1,0,0,0,0,1,1) : 114 : 3 &
\end{array} \right\}$$

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

A. As it comes to our implementation, obviously the answer is no; no matter which channel the transmitted word is going to be observed from, the maximum available number of secret bits in our model is 3, and can take place only in the eavesdropped channel. In order to achieve 4 secret bits over the eavesdropper channel, we should implement the possible injection of an additional binary error. In the case of the legitimate channel instead, we would need to increment the possible injection by 3 additional bits.

This will require obviously re-define the channel_error vectors so that the output of the xor operation will introduce the novel degree of secrecy.

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

A. Yes, as long as it concerns the eavesdropper channel, we find ourselves able to obtain up to 3 secret bits per word already, as discussed in the previous point. In order to get there for the legitimate channel tho, we would still need to provide the possibility for an additional error bit to be injected and the xor output to be compliant.

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?**

A. As we saw, the 3 bit injection drastically harden the security of this mechanism by increasing the output space dimension from 8 to 128. The decoder we implemented on Task 3 works using approximation with a Hamming distance 1. Nevertheless, such a naive implementation in the encoding and error injection will always allow us to easily understand which encoding were transmitted before the noise up to a 3-digits Hamming distance. A better, safer version will obviously require a different Hamming Code, having codes with hamming distances also less than 3. This will mean that the eavesdropper would not be able to infer which codeword was used for encoding. If I still want to make the legitimate channel work, I should create the Hamming code such that its elements have Hamming distance higher or equal to 2. Then I should allow the legitimate channel to inject an error bit. This will still allow to univocally recognize the proper encoding from the legit user. Instead, I should have the eavesdropper noise to be at least up to 2 bits always. Therefore when recostructing the encoding the eavesdropper would always have more possibilities, without being able to infer the right one.

Obviously when dealing to huge error bits injections and bigger spaces, adjusting the threshold over what the legitimate channel should reconstruct easily and what the eavesdropper's should not can be tricky and hard to scale. Moreover all the remarks we discussed where based on the fact that the eavesdropper could gain access on the Hamming-code encoding method, as from [shempfer] assumption Therefore, the best and most general way to test secrecy should always be assessing the mutual information beetween the input message and the observation of the eavesdropper, as the best way to increase secrecy woul be ensuring that the plaintext is not leaving too many useful infomation about itself during the encoding.