Problem 1.1 (Sparse PUFs)**.** Melbo wants to design a super-simple PUF that nevertheless offers good security. The idea makes use of a *conditional delay unit* (CDU) described below. A CDU takes in two inputs, a select bit and a signal. If the select bit is 0, then the CDU relays the input signal to its output without any delay i.e. 0 delay. If the select bit is 1, then the CDU relays the input signal to its output after a delay of p microseconds where p is kept secret and given a certain value of p, it is difficult to create another CDU with that exact value of delay.

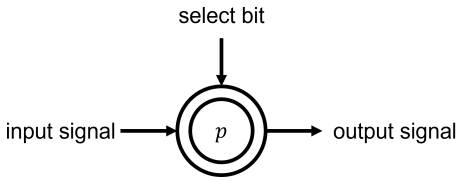

Figure 1: A single conditional delay unit.

CDU PUF: Melbo's idea is to create a good PUF by stringing together several CDUs in sequence. The select bits to these CDUs becomes the challenge and the responses is the total delay incurred, expressed in microseconds. See the figure below explaining delay calculation.

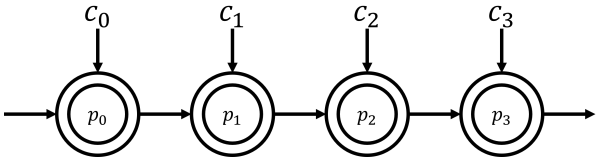

Figure 2: A PUF created by stringing together 4 CDUs. The response is the total delay incurred in the entire chain. For the challenge 0011, the delay would be $p_2 + p_3$ while for the challenge 1010, the delay would be $p_0 + p_2$ and for the challenge 1101, the delay would be $p_0 + p_1 + p_3$.

Sparse CDU PUF: Melbo was happy with this PUF construction till Melbo's sibling Melbi pointed out that a linear model can easily break this PUF as the responses are simply the sum of the delays of the CDUs chosen by the challenge. To make the PUF more challenging, Melbo devised a clever solution of introducing dummy/irrelevant CDUs into the pipeline. See the figure below that explains the trick. Melbo introduces D CDUs but only $S < D$ of them actually participate in the response generation process. The rest $D - S$ CDUs are disconnected from the chain. Thus, the value of the select bit sent to disconnected CDUs has no effect on the response. To make the problem challenging, Meblo refuses to reveal which S CDUs are active.

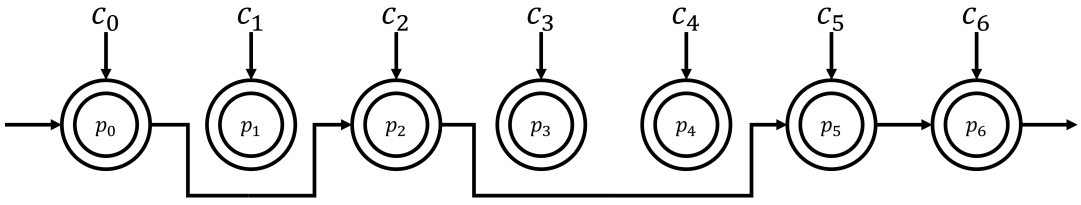

Figure 3: A Sparse CDU PUF with $D = 7$ CDUs out of which only $S = 4$ CDUs are actually participating in the response generation process. Thus, the values of c_1, c_3, c_4 have no effect on the response. For the challenge 1101011, the delay would be $p_0 + p_5 + p_6$ and the same delay would be obtained even for the challenge 1000111. For the challenge 0111010, the delay would be $p_2 + p_5$ and the same delay would be obtained even for the challenge 0110110.

Your Data. We have provided you with data from a Sparse CDU PUF with $D = 2048$ and $S = 512$ i.e. only 512 of the 2048 CDUs are actually active and the rest 1536 are disconnected and play no role in response generation. However, it is not known which 512 CDUs are active. A total of 1600 CRPs have been provided to you. The file `train_challenges.dat` contains 1600 lines each line giving a string of 2048 bits denoting the challenge. The file `train_responses.dat` containing 1600 lines each line giving the response obtained on those challenges in the form of delay in milliseconds. Note that numbers provided in `train_challenges.dat` are bits i.e. either 0 or 1 whereas those in `train_responses.dat` are floating point numbers. Note that the number of training CRPs ($N = 1600$ in this case) was deliberately kept small by Melbo (note that $N \leq D$) to make the problem challenging.

If you wish, you may create (held out/k-fold) validation sets out of this data in any way you like. Your job is to learn a linear model that is D-dimensional but S-sparse that can accurately predict the responses on the test set. Thus, your model should be a single D-dimensional vector of which only S coordinates are non-zero and the rest $D - S$ coordinates should be zero. If you generate a model that has more than S non-zero coordinates, then we will forcibly make your model S-sparse using *hard thresholding*. See the Google Colab evaluation file linked to find out exactly how hard thresholding would be performed.

Your Task. The following enumerates 4 parts to the question. Parts 1,2,4 need to be answered in the PDF file containing your report. Part 3 needs to be answered in the Python file.

1. By giving a detailed mathematical derivation (as given in the lecture slides), show how a sparse CDU PUF can be broken by a single sparse linear model. More specifically, give derivations for a map $\phi : \{0,1\}^D \to \mathbb{R}^D$ mapping D-bit 0/1-valued challenge vectors to D-dimensional feature vectors and show that for any S-sparse CDU PUF, there exists an S-sparse linear model i.e. $\mathbf{w} \in \mathbb{R}^D$ such that $\|\mathbf{w}\|_0 \leq S$ i.e. $\mathbf{w}$ has at most S non-zero coordinates and that for all challenges $\mathbf{c} \in \{0,1\}^D$, the following expression $\mathbf{w}^\top \phi(\mathbf{c})$ gives the correct response. Note that no bias term (not even a hidden one) is allowed in the linear model. (10 marks)

2. Write code to solve this problem by learning a sparse linear model. Note that you may only use the `numpy` library to write your code (e.g. you may use the `lstsq` method to solve least squares – see below for more details). You are not allowed to use any machine learning library e.g. `sklearn, pandas, statsmodels, scipy, tensorflow, keras`, etc. Submit code for your chosen method in `submit.py`. Note that your code will need to implement a method called `my_fit()` that should take CRPs as training data and learn the S-sparse linear model. We will use your provided method to train on the training data we have provided you and then test it on a secret test dataset which consists of CRPs generated from the same Sparse CDU PUF. The same CDUs are active for the secret test set data as were for the train set data. (40 marks)

3. Below are described a few methods you can use to solve this problem. Give an account of which methods you tried (you can try methods other than those mentioned below as well) and why did you like one method over the other. Note that you only need to submit code for the method you found to work the best but in your report you must describe your experiences with at least one other method that you tried. (5 marks)

4. The method you submitted may have hyperparameters such as regularization constants, step lengths etc. Describe in detail how you chose the optimal value of these hyperparameters. For example, if you used grid search, you must show which all values you tried for each hyperparameter and what criterion did you use to choose the best one. (5 marks)