**SAiDL Section 1**

**Step 1: finding a phenomenon and asking a question about it.**

In cryptography, learning parity with noise problem or the LPN problem turns out to build a very hard to crack cryptographic algorithm so much so that the LPN problem cannot be solved using quantum computers i.e. it is quantum-resilient unlike NT based hard problems (RSA/ ElGamal cryptosystem). This text will talk about **where the LPN problem arises in the real world and how some machine learning models will solve the LPN problem.**

Having said that I won’t talk about some deep learning neural network model with several layers sure that would get higher success rates, use fewer samples, and will be able to solve problems with higher dimensions.

**Step 2: Understanding the state of the art**

Let’s stick to one particular cryptographic use case so that I can concretely talk about it throughout this text. Consider a hotel, suppose you are concerned about the lock and key system of doors for hotel rooms.

Now traditionally one could argue about using the age-old physical keys, but the disadvantage of physical keys besides them being boring old-fashioned is that the guest or staff could lose keys and that could cost both security and money to your business, also, in general, they aren’t very secure.

One could think to remedy this problem by deploying smart cards, specifically cards with RFID (radio frequency identification) chips and corresponding locks. Since we are low on the budget we go for a very weak RFID chip, in other words, chips with diminishing computational power. The way this will work is that every lock and every smart card has a secret key, a binary vector say, 96 bits or 128 bits long. Guest holds their card next to the lock which works as a reader, scanning the card’s secret key. If the secret key matches, the door opens.

The caveat here is that a tech-savvy person known as an attacker in crypto lingo can read the communication between lock and guest’s chip when guest is trying to enter their room. The attacker can see the secret key, and forge a card to easily enter the room.

But you’re in no mood to compromise security and the low budget, because of low budget you cannot encrypt anything as the chip has nearly no computational power and also barely enough storage space for the secret key.

You decide to use a cryptographic protocol like HB protocol to trick the attacker. In this, we have a reader R( the lock) and a smart card or Tag T( your chip). T wants to prove to R that it possesses the same secret key without revealing it. The theory is the same as something known as “zero-knowledge proof” in crypto lingo. Anyway, R repeatedly challenges T with questions only a tag with the correct key can answer. In HB protocol T is asked to only reveal small portions of the secret key one tiny bit at a time, until R can be sure that T has the correct secret key.

Say secret key of R and T is in fact same and for simplicity say, s= (1,0,1,0)

R sends random vector a, a = (1,0,1,1) say.

R expects T to return the scalar product b.

i.e. b = <a,s>

i.e. b = <(1,0,1,1), (1,0,1,0)>

i.e. b=0 (remember all calculations are done in F2 field because of binary)

R can calculate the scalar product itself and check T’s answer. The door unlocks after several tests.

If T does not have the correct key since single response would be correct with the probability, p=0.5, hence after say 10 rounds, the chance of successful authentication = 1/(2^10)

= 1/1024

i.e less than 0.1% chance of false positive.

Sadly the attacker can easily crack such a protocol. The attacker would simply track the entire communication between R and T and solve a linear equation to recover s.

Eg.

a1= (1,0,1,1) b1=0

a2= (0,1,1,1) b2=1

a3= (0,0,1,0) b3=1

a4= (0,0,0,1) b4=0

now <ai , s> = bi

ஃ



This won’t take long to yield s, even if it were 1024 bit long. Simple Gaussian elimination.

Bummer.

The learning parity with noise problem:

As it turns out, a small tweak brings a huge change. If T just adds some random Bernoulli noise, e, to its responses, i.e. instead of sending <a,s> back to R if

with probability p, it sends back : <a,s> + e

With probability 1-p, sends back: <a,s> +0

We assume p < 0.5

Attacker still takes account of the entire communication, only this time he has to solve the problem As ≈ b

Because As = b is true only with probability 1-p.

Thus the attacker has to solve a noisy system of equations over F2. For a constant error rate p, this LPN problem is conjectured to be infeasible to solve for a large enough length of the secret key.

[Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. New results for learning noisy parities and halfspaces. In 47th FOCS, pages 563–574, Berkeley, CA, USA, October 21–24, 2006. IEEE Computer Society Press]

R determines whether T knows s or not if a fraction of about 1-p responses will be correct. If p=0.25 and HB protocol runs for 1000 iterations, T should give around 750 correct answers.

If T does not have correct s it gives around correct answer with a probability of 0.5, i.e. 500 out of 1000 times.

**Step 3: determining the basic ingredients**

We now know solving the LPN problem means, given a random binary matrix A and tag returns a binary vector b such that,

b = As + e e = Noise, chosen from Bernoulli Dist. With

"Pr(e = 1) = p where 0 < p < 0.5

now: each row aᵢ of matrix A → **sample** and

the corresponding value bᵢ = <aᵢ, s>+eᵢ in the vector b → **label**

Here, label bᵢ = scalar product of the feature vector aᵢ and a fixed secret vector s (some truth), but with an error term added.

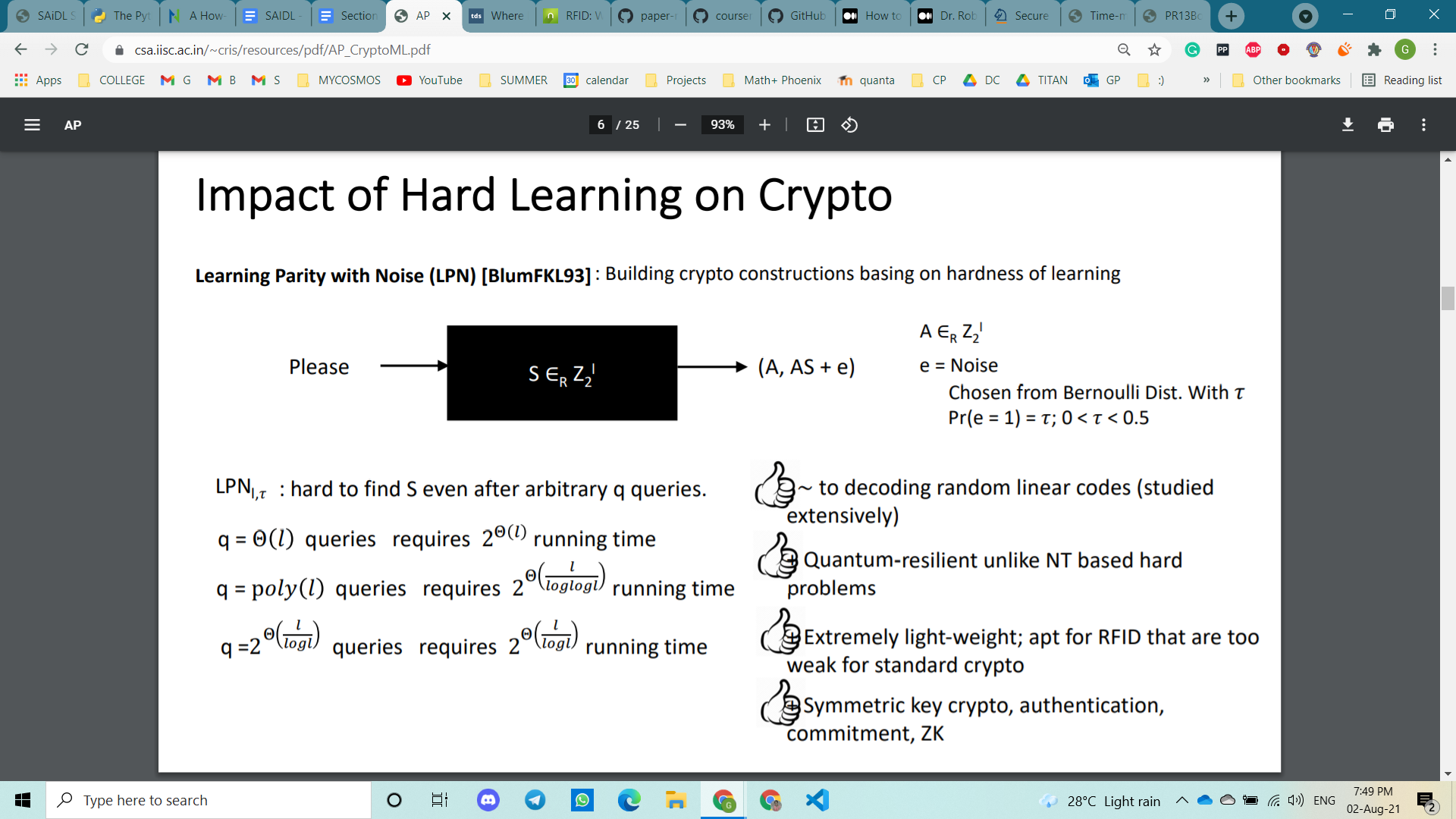
ML idea remains the same, we have some input dataset and we have an objective, in this case, classify the key, is it the right key or not. What we feed the model is a bunch of samples in the form of matrix A and labels b. The idea is that the ml model learns the problem reasonably well.

**Step 4: formulating specific, mathematically defined hypotheses**

So if the ml model learns the problem well, it could generate good predictions for the labels (the scalar product) for each feature vector aᵢ we like. If we throw in the vector a=(1,0,0,0), we would then receive a good guess for



Notice s1 is the first bit of s. Do the same with the vectors (0, 1, 0, 0), (0, 0, 1, 0) and (0, 0, 0, 1) and we have all the bits of the secret key.



**Implementing the model**

**Step 5, Step 6: selecting the toolkit, planning the model**

We will use the **NumPy library** to generate a random matrix A with rows equal to the number of tests the R wants to perform and the number of columns equal to the dimension of the secret key, to add random errors and compute labels.

We try to find s using a decision tree machine learning model using the **sci-kit learn library** in python.

**Step 7, Step 8: implementing the model, completing the model**

[exercise](https://colab.research.google.com/drive/1-DuGmu4z0y8_R7Prl2lIervMo8ipwEum?usp=sharing)

**Step 9: testing and evaluating the model**

The decision tree learning algorithm used here might fail to capture the truth and learn another function. In this case, the count of number of 1’s becomes very high (nearly 50000 for our vector of length 100000 i.e. 0.5 fraction of times). This corresponds to the case where the tag T has the wrong key and can answer about half of the challenges correctly. If s\_candidate == s, i.e. the obtained s-candidate is same as secret key then the count of 1’s should be, 0.125 \* 100000 = 12,500.

Having a threshold of 14,000 is a good tradeoff between recognizing the correct secret and not outputting a wrong candidate as the solution.

The model can only solve a small instance of the secret length, for longer secret length and for better overall performance some neural network model with several hidden layers model could be deployed, higher success rate and the need for fewer samples is a motivating push. Also currently a secret length of 512 and p=0.125, is unbroken.

**References:**

[**https://www.csa.iisc.ac.in/~cris/resources/pdf/AP\_CryptoML.pdf**](https://www.csa.iisc.ac.in/~cris/resources/pdf/AP_CryptoML.pdf)

[**https://towardsdatascience.com/where-machine-learning-meets-cryptography-b4a23ef54c9e**](https://towardsdatascience.com/where-machine-learning-meets-cryptography-b4a23ef54c9e)