

**Computational Number Theory and Algebra**  
**Prof. Nitin Saxena**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology-Kanpur**

**Lecture – 14**

Okay, so many weeks ago we were doing Reed-Solomon code.

(Refer Slide Time: 00:18)

Decoding RS [Peterson 1960]

- Idea: Consider the error-locator  $Q := \prod_{j=1}^t (x - e_j)$ .

$\Rightarrow (c'_j - c_j) \cdot Q(e_j) = 0, \forall 0 \leq j \leq n-1$ .

$\Rightarrow P(e_j) \cdot Q(e_j) = c'_j \cdot Q(e_j), \quad "$

[L.Sys.]  $\Rightarrow R(e_j) = c'_j \cdot Q(e_j), \quad "$  where  $R(x) := P(x) \cdot Q(x)$

$\rightarrow$  Coefficients of  $R \in Q$  are the unknowns.

$\deg = k+1+t \quad \leftarrow \quad \deg = t \text{ \&monic}$

So this was the decoding algorithm.

(Refer Slide Time: 00:21)

Reed-Solomon Code

• View the message as a function:  $\mathbb{F}_q \rightarrow \mathbb{F}_q$ .  
 $\rightarrow$  (univariate polynomial)

A sends its evaluations to B.

Encoding: 1) Break the  $N$ -bit msg  $m$  into  $k$  blocks each of size  $b$ -bits.  
 View these blocks as  $d_0, \dots, d_{k-1} \in \mathbb{F}_{2^b}$ .

2) Define  $P(x) := d_0 + d_1 x + \dots + d_{k-1} x^{k-1} \in \mathbb{F}_{2^b}[x]$ .

3) Pick  $n$  distinct points  $e_0, \dots, e_{n-1} \in \mathbb{F}_{2^b}$ .

Send code  $(c_0, \dots, c_{n-1}) := (P(e_0), \dots, P(e_{n-1})) \in (\mathbb{F}_{2^b})^n$ .

And in the code the encoding was just so you have this big  $N$  many bits that Alice wants to send to Bob. And so you break it into blocks of  $b$  bits, and there are  $k$  many blocks. So  $b$  times  $k$  is equal to  $N$ . And these each block  $b$  bit you see as a finite field

element, right. A finite field of size  $2^b$ . So it naturally is a finite field element and there are  $k$  elements, so you get a degree  $k - 1$  polynomial.

It is a non-monic polynomial, although that thing is not very important. But in degree  $k - 1$  you need  $k$  coefficient. So the message gives you that. And that defines a polynomial  $P(x)$ . You send evaluations of this polynomial. Usually in applications it is assumed that you will send all possible evaluations, okay. So  $2^b$  is the finite field size. You will evaluate your polynomial at each field element.

And you will send these  $2^b$  values. So that is small  $n$ , okay. So you send  $n$  finite field elements, which is basically the polynomial evaluated at all the points. And then arbitrarily this information will get corrupted in the channel and up to some limit I mean if the number of errors is below the threshold then the decoding algorithm will still be able to recover  $P$  and hence the message. So any questions. So you can set the parameter.

(Refer Slide Time: 02:11)

$$\begin{aligned} \Rightarrow \Delta &= R' - P \cdot Q' = 0 \\ \Rightarrow P &= R'(x) / Q'(x) . \quad \square \end{aligned}$$

$$\triangleright \text{Error-tolerance} := t \leq \frac{n-k}{2} = \frac{n}{2} \left(1 - \frac{k}{n}\right)$$

$$- 2^b \geq n \geq k + 2t \quad \& \quad N = b \cdot k .$$

$N$  given parameter

$$\rightarrow k, b, n \text{ could be fixed for any desired } t < n/2 .$$

$$\triangleright \text{Time to solve the Special th. sys} \leq \tilde{O}(nb) .$$

From the decoding algorithm we got error tolerance nearly  $n/2$ , okay. Small  $n$  is the number of values sent and if the error is less than  $n/2$  half of them less than half of them get corrupted, still it will work. Okay which is a very surprising fact because  $n/2$  is the theoretical limit, okay and your algorithm is actually able to correct nearly up to the theoretical limit.

And today we will actually even cross this theoretical limit and we will be able to correct errors beyond  $n$  by 2, okay which sounds impossible but it is doable.

(Refer Slide Time: 03:02)

- Example fixing of parameters:  
 $b = \lg N$ ,  $k = N/\lg N$ ,  $n = N$   
 $\Rightarrow t \leq \frac{n-k}{2} \leq \frac{N}{2} \cdot \left(1 - \frac{1}{\lg N}\right)$   
 $\approx 50\%$  error-correction in  $\mathbb{F}_{2^b}$ -alphabet.  
 $\triangleright$  RS-code is of length  $= N \lg N$  & corrects up to  $\frac{N}{2} \left(1 - \frac{1}{\lg N}\right)$  errors.

But anyway, so here we set the parameters and okay. So big  $N$  is the main parameter, which is it is a single parameter setting that is the number of values sent on the channel. The  $k$  is the number of coefficients. So that is just a little bit smaller. So the stretch here is very small. The stretch is only from  $N$  by  $\log N$  to  $N$ , okay. It is minimal stretch while the brute force would have required  $n$  square.

And the field size is also around  $N$ .  $2$  raised to  $b$  is also around  $N$ , okay. And for that error tolerance  $t$  is this  $N$  by  $2$  times nearly  $1$ . As  $N$  tends to infinity this is nearly  $1$ . So this is nearly  $N$  by  $2$ . So  $50\%$  error correction in the finite field alphabet. So alphabet here is not  $0, 1$  but finite field. So the element we are counting as  $1$  extended bit of information.

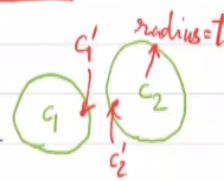
Yeah we did not go into the details but all this can be done in nearly linear time in  $N$ . Okay, the evaluations and then the decoding etc., is quite fast. Okay. So with that we will move forward.

(Refer Slide Time: 04:31)

Distance

- $(2t+1)$  is called distance of the (RS) code.
- It is the minimum Hamming distance between any two distinct codewords.

▷ Distance =  $\Delta \Rightarrow$  error-tolerance (of any code) is  $< \Delta/2$ .



- With error-bound  $t \geq N/2$ , there are many messages corresponding to a corrupted codeword.

So there is one concept the underlying concept of distance in this code and in general in error correcting codes. So for example, if in the previous slide this  $t$  that was error tolerance of the code  $2t + 1$  is called the distance. So in our case you have only seen one example of Reed-Solomon code. Distance of that is  $2t + 1$ ,  $t$  as before.

This is called distance because combinatorial if you look at two code words they are sufficiently far away in the space which means that even if the code word which was sent on the channel even if there is a corruption which is within a neighborhood of the code. From that corrupted string you the decoding algorithm will give you the center of the ball which is the code word and hence the message, okay.

So the balls around code words they are kind of non-overlapping. And their, you can say their radius is  $t$ . How do we define? Right. So it is the minimum hamming distance between any two distinct code words. So in a picture if you have ball around a code word  $c_1$  and around a code word  $c_2$ . So code word is string of letters where in our case the alphabet is finite field element.

So  $c_1$  is basically this big  $N$  many field elements and  $c_2$  is another array of big  $N$  many field elements. So in how many locations are they different, okay? So in this alphabet that is what we are calling hamming distance. Usually it is for binary strings, but the same thing you can generalize to any alphabet. So  $c_1$ ,  $c_2$  are basically two arrays equal length and what is the difference.

How many locations are there where the arrays are different? That is called hamming distance between  $c_1, c_2$  in this alphabet, in the finite field alphabet. And over all the pairs of code words pick the minimum, okay. Minimize over all pairs of  $c_1, c_2$ ;  $c_1$  different from  $c_2$  of course. So that is called basically distance of a code. Yes.

**“Professor - student conversation starts”** Is the distance then twice the whatever error you are decoding? **“Professor - student conversation ends”**. Yeah. So whatever is your yeah whatever is the requirement in practice you want to tolerate let us say  $t$  many errors. Then it is clear that the distance between these two arrays has to be more than  $2t$ , strictly bigger than  $2t$ .

Even if it is  $2t$  you might be in a situation where  $t$  errors happen. And then that will be, that will correspond that corrupted the corrupted code word will correspond to two balls. And then you cannot distinguish how did it arise, what was the message? So the balls really have to be non-overlapping, right. So whatever is your desired error tolerance, distance has to be strictly twice as big. So that is it.

So if you have, now when this code words even is sent on the channel, then this might shift to this position  $c_1'$ , right. And  $c_2$  when sent over a corrupt channel, erroneous channel it may shift here  $c_2'$ . So it might shift here and this might shift here. So  $c_1$  may actually shift closer to the ball  $c_2$ , around  $c_2$ . And  $c_2$  may shift closer to the ball around  $c_1$ . So they are getting very close.

And if these two balls were actually there was an overlapping point that point then you will not be able to decode. Because it will correspond to two code words. But in this case it is okay because the nearest code word to  $c_1'$  is still  $c_1$ , right. So that is the actually the advantage that Bob has. No matter what the channel does within the threshold of  $t$ ,  $t$  or less Bob will be able to always get to  $c_1$  from  $c_1'$ .

And this radius is  $t$  of course of the balls, okay. So up to  $t$  errors, these corrupt strings will remain inside the ball. Any questions? So this is the notion of distance of a code word, of coding scheme in fact, in general. It is clear that you cannot cross 50%. You cannot even achieve 50%. So distance  $\Delta$  implies that error tolerance of any code word, of any coding scheme is less than  $\Delta/2$ , right.

So if you have a mathematical coding scheme with a proven distance of  $\delta$  or less, then in practice the error tolerance will be less than half. It cannot be better than this. So distance can be in terms of fractions it can be at most 1 and so the error is always, tolerance is always less than half, okay. So what in particular what it means is with the error bound.

So if there are half or more errors, the problem that is happening in decoding is that there are more than one possible code words and hence messages corresponding to what Bob got, Bob received. There are many messages corresponding to a corrupted code word. So there are two things now. So we basically want to cross this barrier of  $n$  by 2. So what if the channel makes corruption more than 50%, right.

So theoretically then it is impossible to find the message. But how many messages are possible corresponding to the string that Bob received, right. So if you can count if you can give an upper bound on the count then there will be hope that maybe there is a decoding algorithm that gives a list of messages, okay. So that is called list decoding. That did not exist before.

Even the concept actually came with the Reed, the list decoding of Reed-Solomon code, which we will see now, okay. And so that amazing algorithm will also give an upper bound on the number of messages corresponding to the corrupted string that Bob got, okay. So it is both an algorithm and a combinatorial result. Because otherwise it is not clear how will you count the number of code words corresponding to a string which is heavily corrupted.

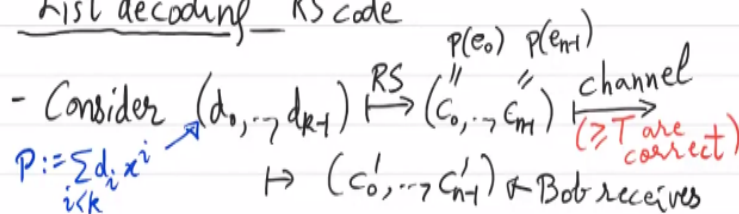
So the corruption is let us say more than 55%. So in that case, how many messages are possible? So the algorithm will not only upper bound the number of search messages, it will also give them to you in the form of a list. So could we find all of them? That is the algorithmic question.

**(Refer Slide Time: 12:57)**

- Could we find the list of messages?

- (Madhu Sudan, 1995) found an efficient way to list decode (RS code).

List decoding - RS code



So in particular are the, is the list small? If the list for example is exponentially long then there is no point finding them, which will happen at some point, right. If the errors are very close to how much? Yeah, so in the extreme case, if every bit has been flipped, almost everything has been flipped, then obviously, you expect the messages to be 2 raised to n many.

The list will be around 2 raised to n. And but what if that has not happened? What if you are still very close to n by 2, somewhere between 50% and 55%? So in that case, is the list still exponentially large, right? So those are the questions of, or how is the asymptotic growing? Because below 50%, it was just 1. So intuitively, beyond 50% it should be slightly bigger than 1 but not exponential, not 2 raised to n.

So that was achieved by a theoretical computer scientist called Madhu Sudan fairly recently. So he found an efficient way to list decode, okay. So even this term was a first. So in particular, list decode Reed-Solomon. So this is what we will see now. So we will do list decoding for Reed-Solomon. So the setting is as before. So just to recall. So you had Alice had a message  $d_0$  to  $d_{k-1}$ .

These are the k field elements. The Reed-Solomon coding algorithm gave her n field elements, which is basically the evaluation. So P is the polynomial that this message defines and the evaluation of that at sorry  $e_0$ .  $e_0$  is the kind of the first field element then  $e_1$  is the second and so on. So compute all these values and just send it over the channel. The channel will corrupt it.

And we will assume that at least  $T$  many are correct, greater than  $\frac{n}{2}$  many. So greater than  $\frac{n}{2}$  are correct. And the remaining  $n - \frac{n}{2}$  maybe are in fact incorrect. They have been corrupted. So what Bob gets is we are calling it  $c_0$  prime to  $c_{n-1}$  prime, okay. So in this only  $T$  positions are correct and obviously, Bob has no idea of which  $\frac{n}{2}$  many positions are these.

There is no way to find that a priori, right because the number of positions is  $\binom{n}{T}$  which is exponentially large. And moreover, we will not restrict  $\frac{n}{2}$  to be half. So the correct ones may be less than  $\frac{n}{2}$  right. So this is a very difficult problem for Bob. So how will you solve it? How will Bob try to find the list of all possible messages  $d_0$  to  $d_{k-1}$ ? For some bound on  $\frac{n}{2}$ , so you fix some bound on  $\frac{n}{2}$ .

Let us say just below  $\frac{n}{2}$ . And within that bound, how many messages are possible, right. That is the question. So the way we decoded, before the unique decoding that was why so there was this auxiliary polynomial, right, error locator polynomial. So what was that? It collected the wrong positions, right. So it was a univariate polynomial whose roots were supposed to be the wrong positions.

So now what is done is something quite different. So now, we will actually error locator polynomial we will pick will be bivariate, okay. So that the idea will be that if the channel was error free then for field element  $e$  the value would have been  $P(e)$ . So the bivariate polynomial will try to interpolate these points, okay. Field element  $e$  comma  $P$  of that element.

So that is the point in the 2D space and the bivariate polynomial will interpolate that. But obviously that is not what is happening in the channel. So the channel will not really correspond to  $P(e)$  for  $n - T$  many places. But anyways, we will build and see what to deduce. So let us first change the error locator polynomial.

**(Refer Slide Time: 20:15)**



- Consider a bivariate "error locator" polynomial  $Q(x, y)$  of degree  $D_x := \deg_x Q$  &  $D_y := \deg_y Q$

(1) s.t.  $Q(e_j, c'_j) = 0, \forall j \in [0, n-1]$ .

[If  $(1+D_x)(1+D_y) \geq n$  then such a  $Q \neq 0$  exists.  
It can be computed by linear algebra.]

- Consider  $R := Q(x, p(x))$ . It has  $\deg \leq D_x + (k-1)D_y$

- We know:  $R(e_j) = 0$  for T-many  $j \in [0, n-1]$   
[by (1)]

$\triangleright T > D_x + (k-1)D_y \geq \deg R \Rightarrow R = 0 \Rightarrow (y-p) | Q$ .

So we are changing the definition. So consider a bivariate error locator  $Q(x, y)$  of degree  $D_x$  and  $D_y$ . These are the individual degrees. So let me define it. So  $D_x$  is the degree of  $Q$  with respect to  $x$ . And  $D_y$  is degree of  $Q$  with respect to  $y$ . Okay, so we call these individual degrees. So now the error locator polynomial will be very different from what we used before.

First of all it is bivariate and in the definition you would not even see anything to do with errors, actually. So we will fix  $D_x$  and  $D_y$  later. Let us first develop the setting such that what we want is  $Q(e_j, c'_j)$ . We want it to be 0. Yeah, so this is just a say a curve that passes through all these points that Bob has received, right. So Bob has received  $e_j, c'_j$ .

So this is just the kind of the curve that fits whatever Bob received. So you do not really see what errors it is capturing, right. The definition is very different from what you saw before. Okay, so for what conditions on  $D_x$  and  $D_y$  will this curve exist? That also we have to remember. It will not exist for every  $D_x, D_y$  right? So intuitively, this  $Q$  will exist only when  $D_x$  and  $D_y$  are sufficiently large.

If one of these is very small, then this low degree  $Q$  will not be able to fit all the points. So exactly the bound you get is this. So as long as  $D_x$ , essentially  $D_x$  times  $D_y$  the product of these two is at least  $n$ .  $Q$  will exist. Why is that? So how many unknowns are there in  $Q$ ? So this  $1 + D_x$  times  $1 + D_y$ . It is a bivariate polynomial of these respective individual degrees.

So that many unknowns and clearly in those unknowns this system is a linear system. So if the as long as the unknowns are more than the constraints. Also observe that this is a homogeneous linear system, right. So there is no question of infeasibility. So just by numbers how many unknowns are there you are guaranteed a solution. So this, essentially  $D_x$  times  $D_y$  should be at least  $n$  and then  $Q$  definitely exists.

In fact many  $Q$ s may exist. Then such a nonzero  $Q$  exists. And it can be computed quite easily by linear algebra, right. So it exists and it can be found easily. So Bob does that. We have still not fixed  $D_x$  and  $D_y$ . That we will, because  $D_x$ ,  $D_y$  we will put more constraints and then we will solve it. Okay, so Bob now has computed this  $Q$ , how do you think this  $Q$  will help?

Do you see that this  $Q$  hides the message, or contains the message or the list of messages, in fact. All possible messages are somehow contained in this  $Q$ . So let us to see that let us look at the idealized setting when the channel does not make a mistake, or does not corrupt. So that would have been, so let us say you look at this univariate polynomial, which is by setting  $y$  equal to  $P_x$ , right?

So a correct so a message  $P$  of our interest. If the channel was correct, then it would have given  $x$ ,  $P_x$ . Bob would have received  $x$ ,  $P_x$ . So if you substitute that in  $Q$ , let us call that polynomial  $R$ . So do you see that this or how can you ensure that the  $Q$  that Bob has found vanishes at  $y$  equal to  $P_x$ , right. This is exactly what you want.

So this bivariate polynomial  $Q$  just computed in step one, what if its factors give you  $y$  minus  $P_s$  for all  $P_s$ , right. So that so basically Bob will then find  $Q$  and then factorize it. And the factors will give the list of messages or in fact, first yeah  $P$  is the message. So Bob will have the list of messages. But for that, we have to or Bob has to make sure that the  $Q$  that has been computed actually vanishes at  $y = P$ .

So this  $R$  should be 0, right? Can that be achieved? Can that be ensured? So let us see the degree of this. So degree is  $D_x$  of  $Q$  plus in  $y$  you have substituted  $P$  whose degree was  $k - 1$ , right. So  $k - 1$  times  $D_y$ . That is the degree of  $R$ . So obviously, we have to use the fact that at least  $T$  many  $e_j$ s are correct, right? So  $R$  at those  $e_j$ s is 0.

So we know that  $R$  at least some of the  $e_j$ s does vanish for  $T$  many  $j$  in  $0 \dots n-1$ , right?

This is the hypothesis. We still have not fixed  $T$ . We have not fixed  $D_x, D_y, T$ . We will do that in the end, but some guarantee has been provided the  $T$  many positions are correct. Sorry? Yeah. So what is  $e_j, P_{e_j}$ ?  $P_{e_j}$  for those  $j$ s is  $c_j$  prime. And the  $Q$  that Bob has found satisfies that. This follows from equation 1, from this equation. So from that equation,  $R_{e_j}$  you can see is  $Q$  at  $e_j, P_{e_j}$ .

But  $P_{e_j}$  is the value that Bob received. So by construction of  $Q$  is true. So for  $T$  many values  $R$  is vanishing. But you actually want  $R$  to vanish for all the values for all  $j$ . I mean, okay not, not really. Or you want just  $R$  to vanish, right? So what matters is what is the degree of  $R$ ? So we have to compare this vanishing of  $R$  at some  $e_j$ s with the degree of  $R$ . So if the degree of  $R$  is less than this, then  $R$  vanishes, right.

That is the, so the last check that you have to do is compare this number  $T$  with the degree of  $R$ . **“Professor - student conversation starts”** But in the first equation we have assumed that it vanishes for all those  $0$  to  $n-1$ , not just for  $T$ . Sorry? In first equation we have assumed that it vanishes for all. **“Professor - student conversation ends”**. That is fine. I mean, how is that in contradiction with, in conflict with the fact that  $R_{e_j}$  is equal to  $T$  many  $j$ ?

You are just using part of the construction of  $Q$ . So the point being made here is that, as long as  $T$  is bigger than the degree, degree is actually smaller than this bound  $D_x + k - 1 D_y$ . So as long as  $T$  is bigger than  $D_x + k - 1 D_y$  right, which is greater than equal to degree of  $R$ . What can you deduce? That  $R$  is 0.

And if  $R$  is 0, then you would have deduced that  $y - P$  is a factor of  $Q$  right, which is what Bob wants that the factors of  $Q$  should contain  $P$ . Because Bob can use enough algebra and come up with a factorization algorithm. You have not seen how to factor bivariate. You have only seen how to factor univariate. But as promised in this course, you will see factorization algorithms for any number of variables.

So Bob can use one of those algorithms factor  $Q$  and hence find  $P$ . And not only one  $P$  but all possible  $P$ s. Moreover, since  $Q$  has limited degree, there are a few  $P$ s, right. So this algebraic root is giving you a very interesting combinatorial fact, right. That the number of such strings that could get corrupted and reach Bob is small. That is the message. That is the  $P(x)$  right. That was the settings we had before.

$P$  is the polynomial you get from  $D$ 's. So  $P$  is  $D \cdot x \cdot i$ . Okay. That is the original message. You want to find the message  $P$ . And since it is not unique, you want to find all of them. So all so right now, we have not fixed anything. So  $T$  is free,  $D \cdot x$  is free,  $D \cdot y$  is free. So you just set them appropriately. And for that setting you have an algorithm. Any questions? So there are yeah, so there is this constraint.

Essentially  $D \cdot x$  times  $D \cdot y$  should be at least  $n$  and there is the other constraint that  $D \cdot x + k \cdot D \cdot y$  should be less than  $T$ . These are opposite constraints. So you have to be just careful to satisfy these opposite constraints. And for that setting of  $T \cdot D \cdot x \cdot D \cdot y$  you have a working algorithm, list decoding algorithm. So let us collect that in a lemma.

**(Refer Slide Time: 34:44)**

Lemma: If  $n < (1+D_x)(1+D_y)$  &  $D_x + (k+1)D_y < T$ , then any curve  $Q$  fitting  $\{(e_j, c_j) \mid j \in [0, n-1]\}$  has  $y - P(x)$  as a factor.

The List Decoding algorithm:

- 1) Fix  $D_x = \sqrt{nk}$ ,  $D_y = \sqrt{n/k}$  &  $T = 2\sqrt{nk}$ .
- 2) Compute  $Q$ :  $Q(e_j, c_j) = 0, \forall 0 \leq j \leq n-1$ .
- 3) Factor  $Q(x, y)$  & collect its factors of the form  $y - f(x)$  with  $\deg f \leq k$ . [ $\#f's \leq D_y \leq \sqrt{n/k}$ .]
- 4) Output the list of  $\{f \text{ as above}\}$ .

So if  $n$  is less than the product and the sum is less than  $T$ . So product should be large and sum should be small, right? If that happens then actually any curve fitting the points that Bob got. This has  $y - P(x)$  as a factor. Okay, so factorization is the way to go. Okay, Bob will just to linear algebra, use a system, linear system solver and then a factorization algorithm and that will give the whole list, okay.

So this is an, it is a first and it is an amazing algorithm. So yeah so the list decoding algorithm let us see with the some parameters fixed. So let us fix, you can see that you want the product large and the sum small. So why not take essentially them to be square root of  $n$ , around square root right. So that is why we will fix something close to square root. So let us fix  $D_x$  to be square root of  $n^k$ .

$D_y$  to be square root of  $n$  by  $k$ . And  $T$  to be twice square root of  $n^k$ , okay. So square roots appear. You can see that  $D_x$  times  $D_y$  is more than  $n$  by design and  $D_x + k D_y$  is smaller than  $T$ , smaller than twice square root of  $n^k$ , right. So those two constraints are satisfied. So you can apply the lemma. I mean obviously, the square root this may not be an integer. So you will just take floor or ceiling.

Simply compute  $Q$  such that  $Q \in \mathbb{Z}[x, y]$  and the degree bounds for  $Q$ , individual degree bounds or  $D_x, D_y$  right. That is the setting. And then third is factorize this bivariate, which you do not know how to factorize right now. But assuming some factorization algorithm factorize this and collect its factors of the form  $y - P(x)$ . So essentially factors that are individual degree 1 with respect to  $y$  and monic, okay.

So these are special factors if you collect them. Let me use different  $f(x, y) - f(x)$ . Is there any property on  $f$ ? Any property of  $f$  that you want to satisfy? How was your message like? Yeah the, so if the if the factor has  $f$  with degree more than  $k - 1$  then it is not a message. So there is no point keeping that in your list, in Bob's list. So these factors with degree of  $f$  less than  $k$ , right.

So you collect these and this is your list of messages  $f$ . So how many are they? What is the upper bound? It is linear in  $y$ . So the yeah  $D_y$ .  $D_y$  is which is square root of  $n$  by  $k$ . **“Professor - student conversation starts”** Yeah, but you are calculating that by degree of  $y$ , right? Yes. And if he wants to look at degree of  $x$  and that will be  $D_x$  by  $k$  because each factor would have a degree  $k$ . **“Professor - student conversation ends”**.

Right. So we can see in the end square root of  $n$  by  $k$ . That is sure. So number of  $f$ 's is less than equal to individual degree of  $y$  which is less than equal to square root of  $n$  by

k, right. So you get a very precise bound, which is which is incredible that, yeah, so let us interpret this. I do not think you have interpreted the parameters, what do they physically mean.

So if you want to send a message with  $K$  field elements by stretching it to  $n$  field elements over the channel, where the channel will at least preserve square root of  $n k$  many locations or field elements correct. Right? So in that case, the number of possible messages corresponding to a corrupted string that Bob received. Number of possible messages is at most square root of  $n$  by  $k$ .

You get a very precise bound for this setting. So how many errors are being tolerated? So that is just  $n - 2 \sqrt{n k}$ . So even in the presence of those many errors, which is actually quite large if you think of  $k$  as much smaller than  $n$ , in that case  $n - 2 \sqrt{n k}$  is nearly close to everything, which is  $n$ , right. It is not really 55% it is more like 99%.

For setting of  $k$ , you can actually make it 99% of the field elements have been corrupted by the channel, right? Only 1% correct information is there. But still a list of all the possible messages will be outputted by this algorithm. Right, but then  $k$  has to be suitably small. So basically what it means smaller  $k$  for the same  $n$  means that you need slightly more stretch that is all. But the stretch that you need is only by a constant factor, right.

So for a constant factor stretch of the message, you can go up to 99% errors, right. This is, this seems theoretically impossible. Okay, so we just output these, that is the final step. So output the list  $f$  as above. That is the list which is output, okay. Any questions about this algorithm?

**(Refer Slide Time: 44:25)**

-lg. setting: For  $n = k \log^2 k$ , we only need  
 $2k \log k$  correct values.  
[Note  $2k \log k / n \rightarrow 0$  !]

▷ This list-decoding algorithm is in randomized  
poly-time.  
It works up to  $(n - 2\sqrt{nk})$  many bit  
errors!

So for  $n$  equal to  $k \log^2 k$ . So basically  $k$  is being stretched to  $k \log^2 k$ , which is not really constant factor. Why did I say constant factor? Okay, no this I think will achieve more. So for this, we only need  $2k \log k$  correct values. Okay, so  $k$  is stretched to  $k \log^2 k$ . But amongst these  $k \log^2 k$  which were sent over the channel, you only need  $k \log k$  values to be correct.

Right, which is actually, if you look at the fraction, this is tending to zero, right. So you really want you just need for this algorithm to work to give a small list the density of correct values is nearly zero. It is not even a constant, right. That is a even more stunning setting. Is this clear? But the stretch here is non constant, it is  $k \log^2 k$  times  $k$ . Okay. Yeah, so what you have in the end is it is collect everything.

So this list decoding algorithm is in randomized polynomial time. Well, we have not shown that completely because of this major missing algorithm to factor  $Q(x, y)$ . But that will be our goal from now. So in the next many lectures, we will see how to factorize bivariate polynomials. Basically, we will reduce bivariate polynomials to univariate polynomials and then use Cantor–Zassenhaus. So everything will be ultimately randomized poly-time.

And it works up to  $n - 2\sqrt{nk}$  many errors. And the list which it will output is very small, square root of  $n$  by  $k$ . So definitely smaller than  $n$ . Yeah, so for this these things to work now we have to solve some new problems, right. So what are the new problems that arise out of this in computational algebra?

**(Refer Slide Time: 48:28)**

- In decoding RS codes we require two new algebraic algorithms:
    - 1) construction of a finite field (eg.  $\mathbb{F}_{2^b}$ ).
    - 2) factoring a bivariate polynomial.
- Constructing  $\mathbb{F}_q$  ( $q = p^b$  given in binary)
- Basically, we want to construct an irreducible polynomial over  $\mathbb{F}_p$  of  $\deg = b$ .
  - We'll show that a random choice works!

So in decoding RS codes, we require two new algebraic algorithms. So first is construction of a finite field itself. In particular, it is  $\mathbb{F}_2$  raised to  $b$ . So we happily said that the encoding algorithm will pick a field and then do the encoding. But since  $b$  is really a variable, how will the algorithm find this finite field or construct this explicitly? So till now you have done many exercises on finite fields.

So you certainly know that they exist and the question boils down to finding an irreducible polynomial of degree  $b$ . But how do you find it? In the application of this coding, encoding algorithm actually  $2$  raised to  $b$  is not large.  $2$  raised to  $b$  is only as big as  $n$ . So you can also do this by brute force, Because the number of irreducible polynomials the space is only  $n$ . So you can just try setting all possible values  $0, 1$ .

But the bigger question is suppose  $b$  is given to you in the input. So then  $2$  raised to  $b$  is exponentially large in terms of  $b$ . So you cannot do brute force. You cannot go over all the polynomials of degree  $b$  basically, the space is too big. So how do you pinpoint, one irreducible out of all polynomials, right. Any ideas? In the first assignment, there was this question where you constructed or you saw how to construct irreducibles over  $\mathbb{Q}$ .

So integral irreducibles, rational irreducibles that there was an explicit example, right. But for finite field, you never saw an explicit example that you can just pick. So this actually is an algorithmic problem. You have to for every input, you have to do some



computations and come up with an irreducible polynomial. So that we will see next. That is the first problem.

The second problem that has arisen is, is a much bigger problem, which is factoring a bivariate polynomial. In fact, if you think about it, given a bivariate polynomial, you currently do not have any idea how to check whether it is reducible or irreducible, right. Forget about factorization. Even a bivariate how do you check its irreducibility, right? That is that seems to be a very challenging question.

So we will solve all those problems at a later point. So what is the easiest thing that comes to your mind to search in this exponential space of polynomials? You already know the key word. No, so you pick a random polynomial.

**“Professor - student conversation starts”** Yeah, pick a random polynomial and factor it. So that is enough? But y is the guarantee irreducible factor would not be necessarily a required degree. **“Professor - student conversation ends”**. No, even better. I am saying that pick a random polynomial. It is already irreducible. It is a one line algorithm. Now if you want to see why it works, well that is of course hard.

So that is what we will do now. Why should such a thing work? So constructing  $F_q$  in general where  $q$  is  $P$  raised to  $b$ . This is given in binary in the input, right. So the input size is  $b \log p$ . So this is simply a number. It is just given to you in the input. And for this number, you have to construct that sized field, finite fields, right. That is the input output description.

You could do it by brute force, but then the time you will spend is  $q$ . And since the input is given in binary  $q$  is exponentially large. When  $b$  crosses 50 then this number is actually more than 2 raised to 50. And that then becomes very quickly, it becomes infeasible to actually go over all degree  $b$  polynomials with coefficients 0 to  $p - 1$ , right. That is a that is truly an exponential space as  $b$  grows.

Basically you want to construct an irreducible polynomial over  $F_p$  of degree  $b$ . So this construction of a finite field is completely equivalent to this irreducible polynomial construction. There is no difference in between these two problems. That

you can see by the exercises on finite field. So finite field will always correspond to an irreducible polynomial. That is the way to represent it.

Then irreducible polynomial will immediately give you a finite field, okay. These two questions are one and the same, also computationally. So what we will show is the random choice works. So random choice will work only if the density of irreducibles is quite high in this space of polynomials, right. So this is basically counting the number of irreducible polynomials.

So we will basically give you the, we will estimate the number of irreducible polynomials of degree  $b$  over  $F_p$ . So have you seen this calculation before? This is sometimes done in discrete math course? Yes. So it yeah for the exact one it does, but I will not need Mobius function. I just want an estimate. So I will approximate everything. What it will use, the starting point is there is this magical polynomial that you have seen before that collects all the irreducible polynomials, right.

So already the irreducible polynomials are collected in one place which is this. Essentially  $x^q - x$ . So this contains degree  $b$  irreducible polynomials but also some lower degree ones. And using the recursive structure there we will get a recursion, which will give us an estimate on the number. Okay, so that Frobenius polynomial is the reason why all this works again.

**(Refer Slide Time: 57:45)**

$$\begin{aligned}
 & - \text{Let } \pi(l) \text{ be the \# irreducibles in } \mathbb{F}_p[x] \text{ of degree } l. \\
 & - \text{Recall that } x^l - x \text{ has, as factors, all irreducibles of } \deg = k \mid l. \\
 & \triangleright p^l = \sum_{k \mid l} \pi(k) \cdot k \\
 & \text{Theorem: } \forall l \geq 1, \quad \frac{1}{2}l \leq \frac{\pi(l)}{p^l} \leq \frac{1}{l} \quad \& \\
 & \quad \pi(l) = \frac{p^l}{l} + O\left(\frac{p^{l/2}}{l}\right).
 \end{aligned}$$

So let  $\pi_l$  be the number of irreducible polynomials in  $\mathbb{F}_p[x]$  of degree  $l$ . Now recall that  $x^p - x$  has as factors all irreducibles of degree  $l$  dividing  $p-1$ , right? That is what we deduced before in Cantor–Zassenhaus analysis or somewhere. So  $x^p - x$  is the product of all those irreducibles when you multiply them, you exactly get  $x^p - x$ , right.

This is without repetition. They are uniquely dividing. Dividing with multiplicity 1. So all your irreducibles of degree  $l$  want to name  $k$  dividing  $l$ . So let us write this as a formula  $x^p - x$  is equal to what?  $x^p - x$  is the degree of this polynomial. It is equal to  $\sum_k \pi_k$  where  $k$  divides  $l$ . And degree  $k$  irreducible has degree  $k$ , right. So and their number is  $\pi_k$ . So this is  $k$  times  $\pi_k$ , right.

So  $k$  times  $\pi_k$  for all  $k$ , their sum is equal to the degree of this Frobenius polynomial. Is that okay? Okay, so just based on this, we will prove a major property that  $\pi_l$  is between, or let me give it as a density. So density, the density of irreducibles amongst all polynomials of degree  $l$  is at most  $1/l$  and  $1/l$  by at least  $1/2l$ . So this is a brilliant result for many reasons.

One is that it actually is an algebraic version of the prime number theorem, right. So prime number theorem also says that the density of prime numbers below  $x$  is around  $1/\log x$ . And what is  $\log$  in this? The analog of  $\log$  here is  $l$ , right. So here also it is saying that it is  $1/l$ . So algebra and number theory are not so different, but algebra is easier, right. That is what you will see.

So this thing has a very short proof. On the other hand, the original prime number theorem has a very involved proof. Yeah, this can be written also in a more precise way. So but that we do not need for the algorithm. It is time is up. So in fact, you can say that  $\pi_l$  is  $p^{1/l} + O(p^{1/2l})$ . That is the main term plus square root of this. Okay, so this second thing is very precise. It is actually giving you the asymptotic behavior of  $\pi_l$ .

The main term is  $p^{1/l}$ . The error term is square root of that. So it is error term is significantly smaller than the main term. Now if you look at the analog of this from for prime numbers, that is still an open question. Okay, whether the prime

number theorem the error term can be made square root. That is the open question for around two centuries. It is called the Riemann hypothesis, okay.

So in one paragraph we can prove Riemann hypothesis here, but not in the prime number, actual prime numbers. **“Professor - student conversation starts”** Prime number is the left hand side inequality valid and 1 by 2l. **“Professor - student conversation ends”**. Yeah, it is some constant. Yeah, if you look at the best results, even this must be true. Yeah, so this is true for all l.

The prime number estimate you need then bigger and bigger x. I remember definitely this on the right side or factor of  $2n$  on the left the factor of half will work in the prime number case. Okay, so we will start this next time. So short proof, it just follows the recursive formula we have written above.

Okay, the above can be seen as a relationship between  $\pi(l)$  and things smaller than l. Like 1 by 2, 1 by 3. So it is a recursive formula. If you use that recursive formula carefully, you can prove this theorem. It is quite easy. Okay.