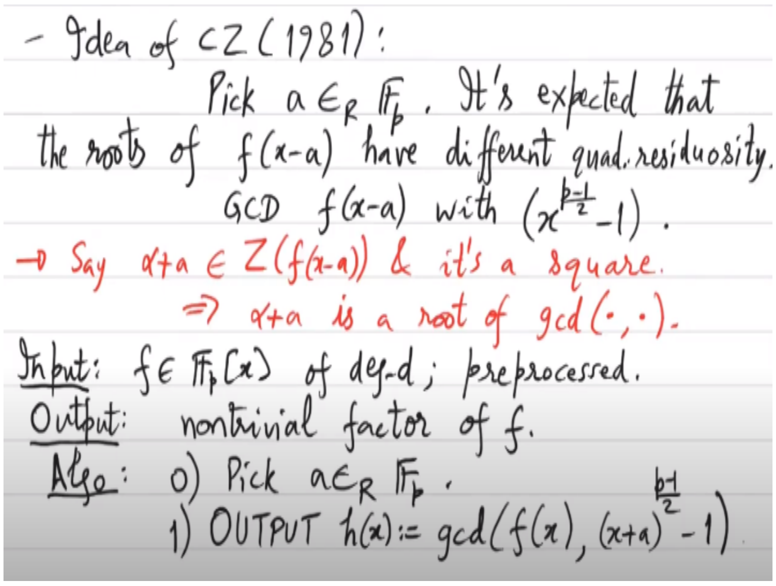
Computational Number Theory and Algebra
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture – 13

(Refer Slide Time: 00:17)



- Idea of CZ (1981):

Pick $a \in_R \mathbb{F}_p$. It's expected that the roots of $f(x-a)$ have different quad. residuosity.

GCD $f(x-a)$ with $(x^{\frac{p-1}{2}}-1)$.

$\rightarrow$ Say $\alpha+a \in Z(f(x-a))$ & it's a square.

$\Rightarrow$ $\alpha+a$ is a root of $gcd(\cdot,\cdot)$.

Input: $f \in \mathbb{F}_p[x]$ of deg-d; preprocessed.

Output: nontrivial factor of f.

Algo: 0) Pick $a \in_R \mathbb{F}_p$.

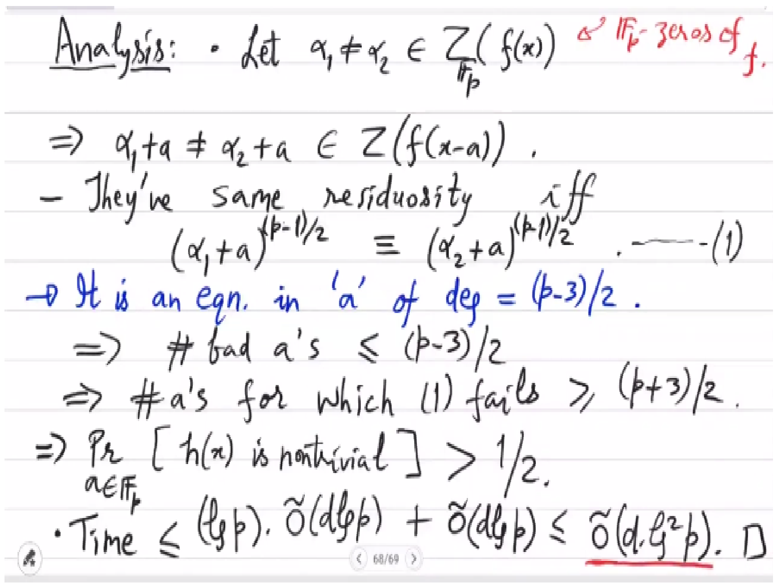
1) OUTPUT $h(x) := gcd(f(x), (x+a)^{\frac{p-1}{2}}-1)$

Okay, so last time we finished Cantor–Zassenhaus algorithm. So in the input, you are given a univariate polynomial over a prime field F p of degree d. So it is preprocessed, which means that F completely split into splits into roots. There are d distinct roots. All are numbers from 0 to p – 1. So the in the output you want to find some factor of f.

So obviously, if you can find one factor of f then repeatedly, you can ultimately find the, apply this repeatedly, ultimately find a root. So Cantor–Zassenhaus algorithm, CZ algorithm is you pick a random a and then just compute this GCD, okay? GCD of f with this shifted x to the p - 1 by 2 - 1. So shift this by a and then compute the GCD.

And the proof was just that roots alpha 1, alpha 2 of f when perturbed by a will have opposite residuosity. So 1 will be a root of, alpha 1 will become a root of h and alpha 2 will not, right. So they will get separated.

(Refer Slide Time: 01:38)


Analysis: • Let $\alpha_1 \neq \alpha_2 \in Z_{\mathbb{F}_p}(f(x))$ α's $\mathbb{F}_p$-zeros of f.

$\Rightarrow \alpha_1 + a \neq \alpha_2 + a \in Z(f(x-a))$.

– They've same residuosity iff $(\alpha_1+a)^{(p-1)/2} \equiv (\alpha_2+a)^{(p-1)/2}$. ------(1)

→ It is an eqn. in 'a' of deg $= (p-3)/2$.

$\Rightarrow$ # bad a's $\leq (p-3)/2$

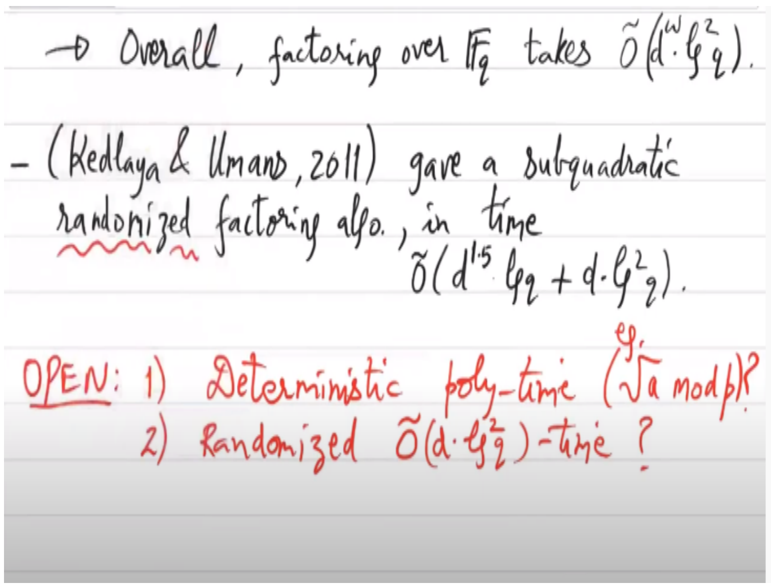
$\Rightarrow$ # a's for which (1) fails $\geq (p+3)/2$.

$\Rightarrow \Pr_{a \in \mathbb{F}_p}[h(x) \text{ is nontrivial}] > 1/2$.

• Time $\leq (\lg p) \cdot \tilde{O}(d \lg p) + \tilde{O}(d \lg p) \leq \underline{\tilde{O}(d \lg^2 p)}$. $\square$

Time complexity is linear. It is this d log square p. But then this is after preprocessing.

(Refer Slide Time: 01:54)


→ Overall, factoring over $\mathbb{F}_q$ takes $\tilde{O}(d^{\omega} \cdot \lg^2 q)$.

– (Kedlaya & Umans, 2011) gave a subquadratic randomized factoring algo., in time $\tilde{O}(d^{1.5} \lg q + d \cdot \lg^2 q)$.

OPEN: 1) Deterministic poly-time (eg. $\sqrt{a} \bmod p$)?

2) Randomized $\tilde{O}(d \cdot \lg^2 q)$-time ?

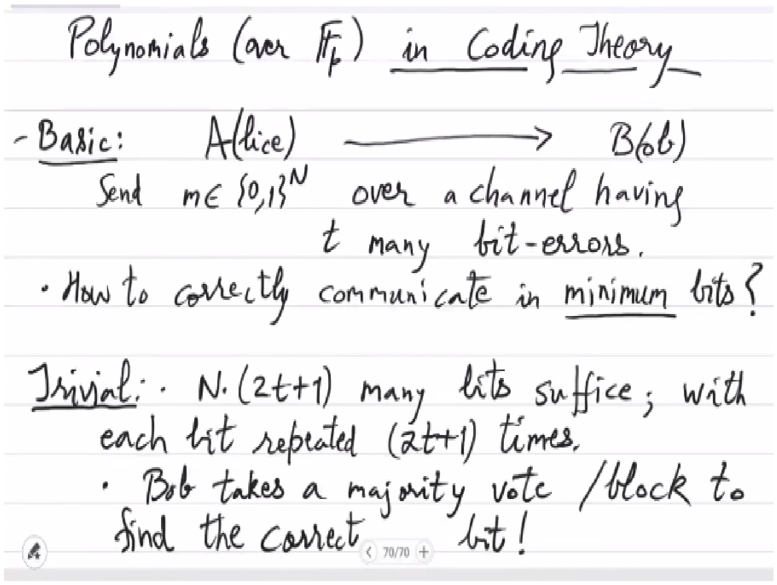
Include preprocessing etc., then overall we have solved factorization over any finite field in sub cubic time. And the best that is known is sub quadratic. Doing it in linear time is an open question from scratch, okay. And all these algorithms fast algorithms that are used in practice, they are all randomized, okay. So another open question is doing it deterministically.

So those are the two open questions. So deterministic poly-time. That is an open question. Second is, so actually even if you qualify this finding square root mod p. So even if you qualify this to degree 2, d = 2, which is nothing but computing square roots modulo a prime, that also we do not know anything better than CZ. Okay, deterministic poly-time algorithms are open. And second is, second is randomized O tilde d time.

Okay, so both these are open questions. The second question is for finite field F q. If q is prime then you already have this solution. Well that was log square p. So maybe I can put log square here. But can you make d linear? Okay, so for over finite fields this is not known. Okay, any questions till now? So with this you can solve factorization. There are no practical difficulties.

Let us move on to the applications of factorization, right. So whatever factorization algorithm we will learn there will be applications. So for finite field there are huge number of applications and then when we will factorize integral polynomials then that also will have huge number of applications. So we will cover some applications also at an abstract level. But it will give you the idea.

(Refer Slide Time: 04:55)



Polynomials (over $\mathbb{F}_p$) <u>in Coding Theory</u>

- <u>Basic:</u> A(lice) $\longrightarrow$ B(ob)
  Send $m \in \{0,1\}^N$ over a channel having t many bit-errors.
  - How to correctly communicate in <u>minimum</u> bits?

<u>Trivial:</u>
- $N.(2t+1)$ many bits suffice; with each bit repeated $(2t+1)$ times.
- Bob takes a majority vote /block to find the correct bit!

So polynomials over finite fields in coding theory, okay. So what is the question in coding theory? So in the first class, this was discussed very briefly that the basic problem is so Alice wants to send so Alice wants to send let us say n bits. So a message that is n bits long to Bob. So the space here is 0, 1 to the big N. Okay, so this message she wants to send over a so it is a physical channel.

So any physical channel will be prone to mistakes, to over a channel having t bit errors, okay. So I mean if she just sends these big N bits the message as plain text then even if a single bit gets corrupted it will be useless for Bob. And Bob will not be and b will not be able to find out which bit is corrupted of course, right. The location is also will also be unknown.

So if it was a English text then Bob can obviously do some error correction by intuition. But if suppose this was a password or this was a number then there is no possibility of any intuitive error correction, right. So even if one bit is lost the whole message is lost. And actually the errors may be in t bits. And t may be actually quite dense in N. So it is possible that t is one-third of N.

So one-third of the bits are actually corrupted and also you do not know which ones. So what is the simple solution for this? Sorry? Parity? So parity is just one bit. But the yeah, you have to pad. You have to basically copy the bits. So you for every bit, so if suppose the error is possible in only one bit. So how much of copying would be required? Right, so at least two and maybe three times, right.

So every bit you have to I mean, if you just repeat a bit twice that still is not enough to know what it was before. So 0, 0 or 1, 1 both of them may become 0, 1. And then from 0, 1 it is never really clear whether original thing was 0, 0 or 1, 1, which means 0 or 1. So you actually need three. If you make it three times copying, then you can take a majority vote. So majority vote will tell you.

So for one bit error, you have to send 3N. And for t bit error yeah for so you have to send 2t + 1 for every bit. So this is really multiplicative. And if t is a if the errors are dense, then it actually becomes quadratic growth. So for to send N bits, Alice essentially has to send N square, right. So that is the brute force solution. So how to correctly communicate in minimum bits.

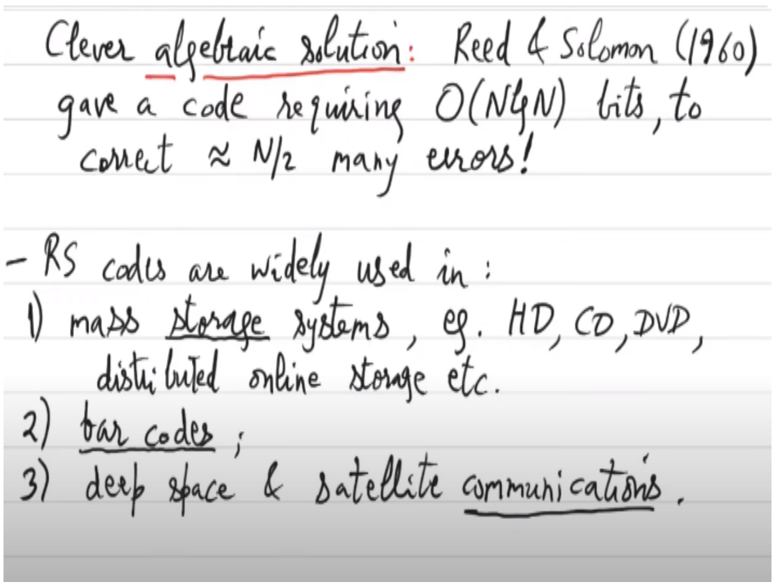
So you want to avoid redundancy as much as possible. Your optimization problem is to minimize the bits sent. So trivial solution is N times 2t + 1 many bits with each bit repeated 2t + 1 many times. And so in this case, the encoding is just copying. And the decoding for Bob is just majority vote. **“Professor - student conversation starts”** Sir, why is it 2t + 1? Yeah, because of majority vote. **“Professor - student conversation ends”.**

So Bob takes a majority vote per block to find the bit, okay. 2t + 1 t may get corrupted, but still t + 1 will be correct. So if you take a majority vote majority will give you the answer. Any questions? So that is, t we will always think of as a fraction of N. So this is really quadratic blow up in the message.

Quadratic blow up actually, maybe completely infeasible. If you are sending, let us say, a satellite is sending images of celestial bodies to Earth, then there is simply no way that you can blow up the image to a quadratic size, okay. That will not be feasible. Because all the instruments involved are small and the distances are big. So you actually want this to be nearly linear blow up, okay.

Quadratic may be actually completely infeasible in this regime. So just saying polynomial in N is not enough, okay? You actually want O tilde N in this case. So what is the solution?

(Refer Slide Time: 12:26)



Clever algebraic solution: Reed & Solomon (1960) gave a code requiring $O(N\lg N)$ bits, to correct $\approx N/2$ many errors!

- RS codes are widely used in:
1) mass storage systems, eg. HD, CD, DVD, distributed online storage etc.
2) bar codes;
3) deep space & satellite communications.

So we will see many or at least one clever algebraic solution. So in all these practical solutions, finite fields are involved. So these are algebraic solutions. So that is due to Reed and Solomon. So they gave a code requiring merely N log N bits to correct nearly half of the errors. Okay, which is a very which is like the optimal solution because errors if the errors are half N by 2 many bits are flipped, then it is information theoretically impossible to recover the message, right.

So just below N by 2, any fraction below N by 2 below half. If it is if the channel is good enough to guarantee that say 0.49 or 49% of the bits will be corrupted. At least 51% will not be, will not be corrupted. If that currently the channel gives you then Reed Solomon code only requires N log N length code word okay. So the code the message is encoded. It is sent over the channel. 49% of the bits of the code get corrupted.

Obviously, the positions are unknown. But Bob has a decoding algorithm which will recover the message. And everything is efficient and optimal. **"Professor - student conversation starts"** Sir, in the channel when we send N bits then the error was of t. Suppose it is N by 3. So now if we are saying 2N bits would the error still be N by 3 or it will become 2N by 3? **"Professor - student conversation ends".**

No, we think of it as a percentage. It is a density guarantee. So basically every bit has a probability of getting corrupted. In practice, it may also depend on the length of the message, but the simplest model is that every bit has an attached probability of getting corrupted. So the basically the expected number of corrupted bits is just probability times the length.

"Professor - student conversation starts" Sir, the previous solution was also like probabilistic, right? It may be possible that all the bits of like 1 log are corrupted. So we cannot get the message. **"Professor - student conversation ends".** Yeah, but the probability will be low. I mean, there is always a probability that your hard disk in your machine will crash, right.

But still you go around every day working on it. So that is all. So everything is probabilistic. Yeah, till few years pass and then your hard disk really crashes and lose everything. So obviously, at that point, no error correction, no code can work. But the probability of that happening is we want to reduce it as much as possible. But still you cannot make it zero. So you can only make probability small, but still it will be positive.

So these coding methods will be required. Without codes computers cannot exist. So this was in fact the first problem to be solved before real computers were made. Yeah, so that obviously gives you all these application possibilities. So Reed-Solomon codes are widely used. Probably their variants and more optimization happens in practice. But the basic idea is from Reed - Solomon.

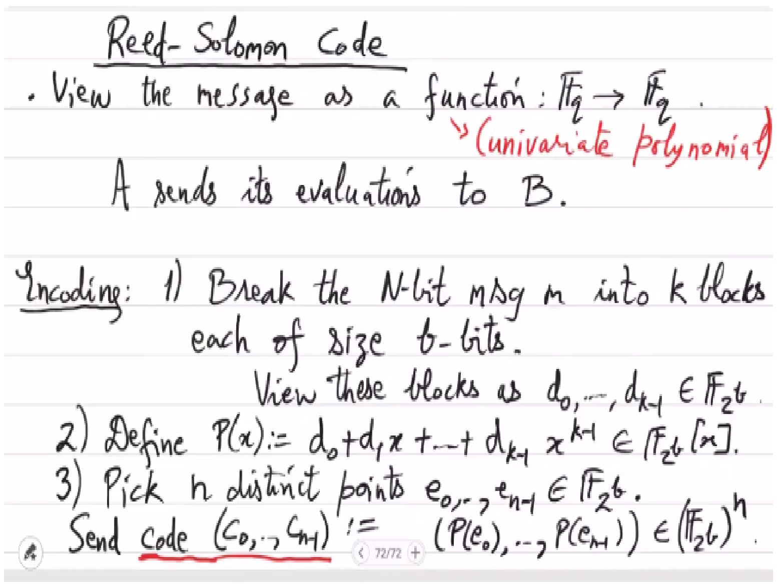
So you will need this for mass storage systems. So like I said, your hard disk and then secondary storage like CD, DVD etc. So distributed online storage. So everywhere you will need some coding mechanism so that even if part of your physical device gets corrupted up to some probability of course, you can recover. When that when a situation happens when the probability when the corruption is more than the guaranteed probability.

Obviously, at that point, you say that your device has crashed, because then your data is not cannot be recovered by using by whatever coding mechanism there was. Okay, so that is the point when you say that your device has crashed. Other things like barcodes. So when you see these barcodes on products, barcodes can also get scratched and bad things may happen to it, but still you want the barcode to work as much as possible.

So even to draw the barcode you need some coding mechanism. And then obviously, all kinds of communication. So to take stunning examples, deep space and satellite communications. But it also includes obviously mundane communication in your cell phone and all. So all this 2G, 3G, 4G, 5G is part of the problem they are solving is improving the coding. So they keep improving it by an incremental amount with every version.

So communications storage and other kinds of places where you need physical sensors, okay. So yeah so let us see the idea of Reed-Solomon, what is the algebraic idea.

(Refer Slide Time: 20:04)


Reed-Solomon Code

• View the message as a function: $\mathbb{F}_q \to \mathbb{F}_q$.
↳ (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 $(c_0, \dots, c_{n-1}) := (P(e_0), \dots, P(e_{n-1})) \in (\mathbb{F}_{2^b})^n$.

So it views them, basically it divides the message into blocks. And then using the blocks as finite field elements, it defines a polynomial, okay. So every message that Alice sends is actually a polynomial in one variable over a finite field. So you can think of every message as a function over a finite field. So it is a function from F q to F q, okay. And to make things keep things simple, obviously, this function will be a univariate polynomial.

But there are variants of Reed-Solomon where this function may be multivariate, okay. So that is called Reed-Muller, etc. Probably, we will not see that in this course. But it is an extension of this basic idea of Reed-Solomon. So this is a univariate polynomial over a finite field. And so what do you want to send over the channel? If you had sent the polynomial itself, then that is basically sending the plaintext, right?

That would not help. So instead, what should you send? No sending coefficients is plaintext. Yeah, so you use the other representation, which is what we also did for polynomial multiplication that you send, you compute the polynomial, and you send the function values. So if you send a lot of function values, even if some of them get corrupted, maybe Bob can devise a clever algorithm to recover the actual function, okay.

So that is the very basic idea of error correction that by looking at some corrupted list of evaluations where some of them let us say one-third of them are corrupted, still you want to learn the actual correct function, okay. And obviously, you do not know which values are corrupted. So Bob will only see a list of evaluations and we will also know the argument.

So Bob sees the argument and the value, but the values may be corrupted, one-third of them in an arbitrary way. From that the function has to be learnt. Yeah, so it is not very easy to see at this point, why this thing is possible, okay. So we will go through this slowly and send the evaluations. So A sends its evaluations to Bob, okay.

So the encoding is, exact implementation will be you first, as I said, consider your message into blocks to get the coefficients of the polynomial. So break the big N bit message m into k blocks each of size b bits. So how will you view these b bits algebraically? 2 raised to b? Yeah so yes so 2 raised to b is the field you pick. You pick the Galois field via some model of size 2 raised to b, which is equivalent to basically picking a degree be irreducible, right.

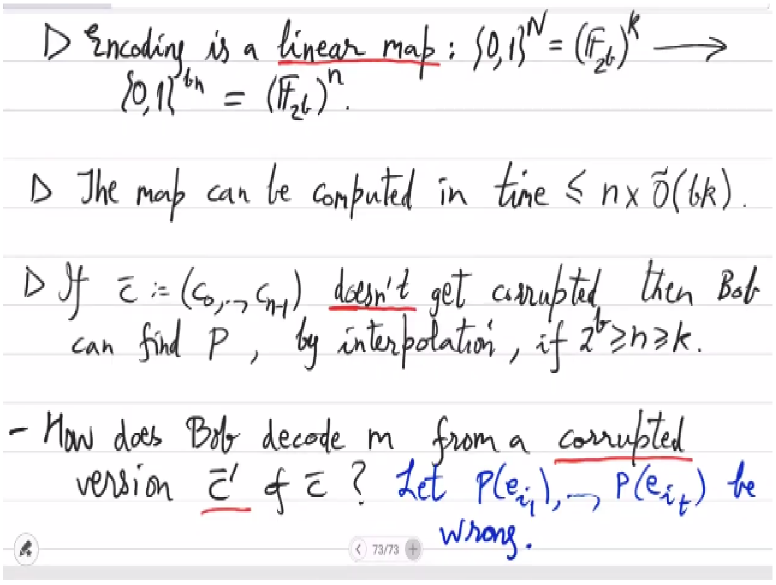
So that will give you a finite field. So you think of these blocks as finite field elements in G F 2 raised to b. And they are k many. So you can build a k -1 degree polynomial. So that is the that function is the message. As elements d 0 dot dot d k - 1 in the finite field of size 2 raised to b. Based on that you can define your polynomial. So your function is this. So it is a univariate polynomial over G F 2 raised to b.

And then you want to evaluate it. So you pick let us say small n many points e 0 dot dot e n – 1 in the finite field. So obviously, when I say that it is meant that n has to be less than equal to 2 raised to b. Okay, so you should pick this so we have three parameters already. We have B which determines k. And B should be such that 2 raised to b should be more than N. The finite field should be big enough.

So you pick enough points e 0 to e n - 1in the finite field and then evaluate the function and send the values which is P e 0, P e n – 1. Okay, so values are also elements in the same finite field. And they are n many. So that is the definition of the code, okay. So this will be how many bits? B n right? So B small n, B times small n. So obviously, this B times small n will be sufficiently larger than big N.

There will be some stretch. So code actually is stretching the message, but not by much. We will see that later. So this stretched code or string obtained from the message is then sent over the channel. Many of its bits will get corrupted, which means when you look at the blocks many of these P e i's will be wrong. I mean wrongly received at Bob's end. But whatever Bob receives from that Bob has to learn P, right. That is the setting.

(Refer Slide Time: 29:38)



▷ Encoding is a linear map: $\{0,1\}^N = (\mathbb{F}_{2^b})^k \longrightarrow \{0,1\}^{bn} = (\mathbb{F}_{2^b})^n$.

▷ The map can be computed in time $\leq n \times \tilde{O}(bk)$.

▷ If $\bar{c} := (c_0, \dots, c_{n-1})$ doesn't get corrupted then Bob can find P, by interpolation, if $2^b \geq n \geq k$.

- How does Bob decode m from a corrupted version $\bar{c}' \neq \bar{c}$? Let $P(e_{i_1}), \dots, P(e_{i_t})$ be wrong.

So this encoding is a, first of all it is a linear map from 0, 1 to the N viewed as finite field space to 0, 1 to the b n viewed as the same finite field space but slightly bigger, okay. So this is what we are doing. So this 0, 1 to the big N was slightly smaller space. So these elements, we are actually expanding them into a bigger space, finite field space. And then the hope is that even if some of these images get corrupted, still there will be enough information left for Bob to learn.

So this map can be computed in how much time? What do you think? It is already you are doing it little n many times. And how much time will it take to compute P at e i? P was a degree k polynomial, right? Sorry, B k. So yeah time is not much, it is barely quadratic, actually. Small n times B is the output size. And that is being multiplied merely by this k. So it is slightly more than linear, but seems sub quadratic in the output size.

Okay, so it is a fast algorithm. There is no problem in efficiency. So one simple observation is if there are no errors in the channel does not get corrupted then Bob can find P in what case? So suppose you send this c bar, so Alice sends the code c bar over the channel, but the corruption did not happen. So Bob, can try to learn P by interpolation, right. So what are the conditions for that for the interpolation to work?

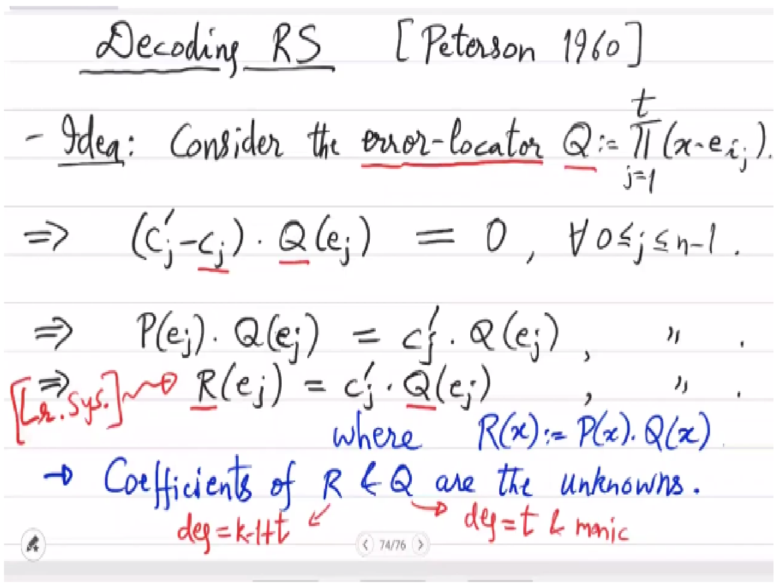
N should be more than more than k, greater than equal to k. So by interpolation, as long as n is at least k and these small n many elements should be distinct, right. So the field size should be at least that. So that is the basic setting you definitely need. So 2 raised to b should be at least n and n should be at least k for any of this to work, because this is needed even when there is no corruption.

So that is the kind of information theoretic bound you have to satisfy. Yeah, but so what if t of these actually got corrupted? What happens then? So this interpolation will completely fail, right. Because interpolation is a very fixed equation. And in that equation, if even a single bit is wrong, then the whole calculation is wrong, right. That does not seem very tolerant to even few errors, even a single bit error.

So you have to come up with a interpolation method that is tolerant to errors. So how does Bob decode m from a corrupted version c bar prime of c bar? Okay, so Bob received some version c bar prime. So let the values which are wrong are e i 1 to e i t okay. So we assume exactly t errors in these unknown places i 1 to i t, okay. And remaining n - t we assume them to be correct.

So this is a thought experiment, because Bob does not know i 1 to i t. In fact, Bob does not even know a t for sure, right. So t is only an upper bound. Errors maybe much fewer than that. So Bob will do a thought experiment first. So suppose that the errors are in these places e i 1 to e i t and try to come up with a method that does not require i 1 to i t in the algorithm, decoding algorithm. So let us title decoding RS.

(Refer Slide Time: 36:48)



Decoding RS [Peterson 1960]

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

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

$\Rightarrow P(e_j) \cdot Q(e_j) = c_j' \cdot Q(e_j)$, " .

[Lin. Sys.] $\rightsquigarrow R(e_j) = c_j' \cdot Q(e_j)$, " .

where $R(x) := P(x) \cdot Q(x)$

$\to$ Coefficients of R & Q are the unknowns.

deg = k-1+t deg = t & monic

74/76

So this mainly is an idea from Peterson. It is a very clever idea. It has remarkable consequences. So the idea is to consider an error locator polynomial. So this Q is basically for Bob it is unknown. It is he calls it an error locator polynomial Q x whose roots are exactly the places where there is a mistake. And observes that the coordinate of corrupted code word jth coordinate.

If you take the product of the difference in what Bob should have received and what Bob actually received times the value of the error locator at the jth point, what can you say about this product? This is always zero, right? Does not matter what j. So for all points, this product is zero. Note that e j is known, c j prime is known, but c j is unknown and Q is unknown right in this equation.

So c j and Q are the unknowns here. So this is the starting equation right and we will just, so Peterson's idea is just to play with this equation and it will give you a decoding algorithm. So this is so what is c j? c j is P e j. So P times Q is equal to c j prime times Q. And let us define P times Q to be R. So R is the product polynomial. So P is anyways unknown. P and Q both are unknown.

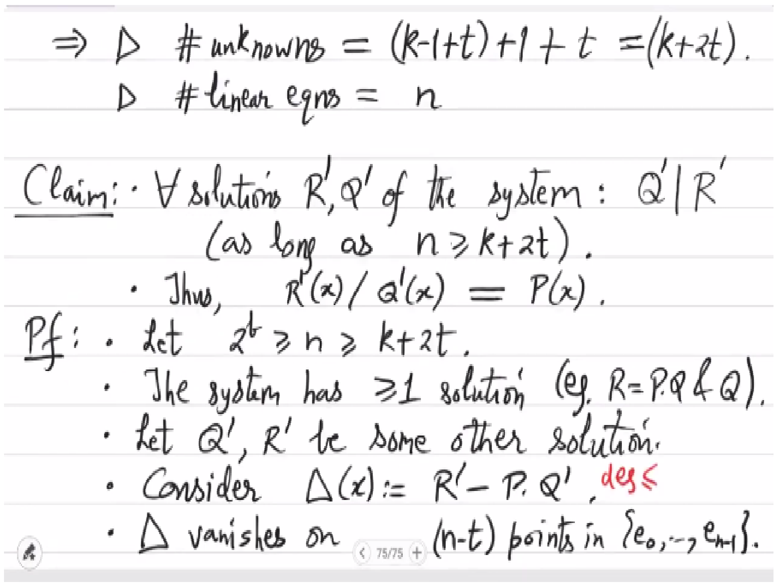
So we just take the product and that R is also unknown. So there is this unknown polynomial R which at value e j is c j prime which is known times Q e j where Q is unknown right. So here R and Q are both unknown. Everything else in this equation is known to Bob. Sorry? E j are the points from the finite field at which the evaluations are taken. These are the arguments.

So in terms of the coefficients of R and Q, what can you say about this equation? What type of an equation is this in terms of the unknowns, what can you say about these equations? Is it linear, nonlinear? Yeah, so obviously, this is a linear system for the unknowns, right. So whenever you see a linear system you solve it. So that is what Bob will do. **"Professor - student conversation starts"** The root of Q are the corrected elements of. Yes that is the error locator yeah. **"Professor - student conversation ends".**

And R is the product of correct polynomial times error locator. So coefficients of R and Q are the unknowns. And in that you have N linear equations, right. So this linear system you will now try to or Bob will try to solve. Okay, let us write down one more thing here. What is the degree of Q? That is t and what is the degree of R? k – 1 + t. R is well Q is monic.

Monic means that the leading monomial has coefficient 1 because it was just the product of x minus e i j's. On the other hand R is not monic, right? So yeah so just remember that, we will use it next.

(Refer Slide Time: 43:35)



$\Rightarrow$ ▷ # unknowns $= (k-1+t)+1+t = (k+2t)$.

▷ # linear eqns $= n$

Claim: • $\forall$ solutions R', Q' of the system: $Q' \mid R'$ (as long as $n \geq k+2t$).

• Thus, $R'(x)/Q'(x) = P(x)$.

Pf: • Let $2^t \geq n \geq k+2t$.

• The system has ≥ 1 solution (eg. $R = P.Q$ & Q).

• Let Q', R' be some other solution.

• Consider $\Delta(x) := R' - P.Q'$, deg $\leq$

• Δ vanishes on $(n-t)$ points in $\{e_0, \ldots, e_{n-1}\}$.

75/75

So this means that number of unknowns is? So for R you get (k – 1 + t) + 1 right because it was not monic, degree plus one. And for Q you get, you exactly get t. So that is k + 2t. That is the number of unknowns. And the number of equations is n, small n. Okay, right. So intuitively, if you have enough equations, then the system then the roots should get or the solution should get determined, right.

So if you take small n to be at least k +2t then it should completely specify Q and R, right. That is the intuition. I mean obviously, whether it will happen or not, that we have to prove. So we will prove this fact that for all solutions R, Q of the system, Q divides R as long as n is sufficiently large, okay. So what is the system? The system is this right? So R e j is equal to c j prime times Q e j for j 0 to n – 1.

This is a linear system with n constraints and k +2t many unknowns. So as long as the constraints are many compared to the unknowns we will prove that whatever solution you find R and Q, Q will divide R, okay. That is a very strong property. Is the statement clear? No. R and Q are arbitrary solutions of the linear system. Yeah, maybe I should use something else here. Maybe R tilde or.

So we are talking about, this is a statement about arbitrary solutions of the same system. Previous slide only tells you that the that system has R solution where R divides Q. But this claim is saying that actually every solution will satisfy this. So that is a very strong claim and it will have a great consequence which is when you compute the quotient R tilde x divided by Q tilde what will that be?

So intuitively that will be the P x. This also we have to show. So basically what Bob will do is that Bob will just find an arbitrary solution of the system and then divide. Then compute the quotient R tilde by Q tilde. That will be the original message, okay. And we will see that this will be a very robust way to interpolate even in the presence of t many errors where t will be quite large.

So the errors are dense, but still this interpolation is robust, okay. It will give you P. Okay, so I think we can do this. So what is the proof? So let us assume that 2 raised to b is at least n and n is at least k + 2t. We are in this setting. So you know that the system has at least one solution. Example R equal to P Q and Q the error locator, right. So the error locator and error locator times original function that is a kind of by definition a solution of this linear system.

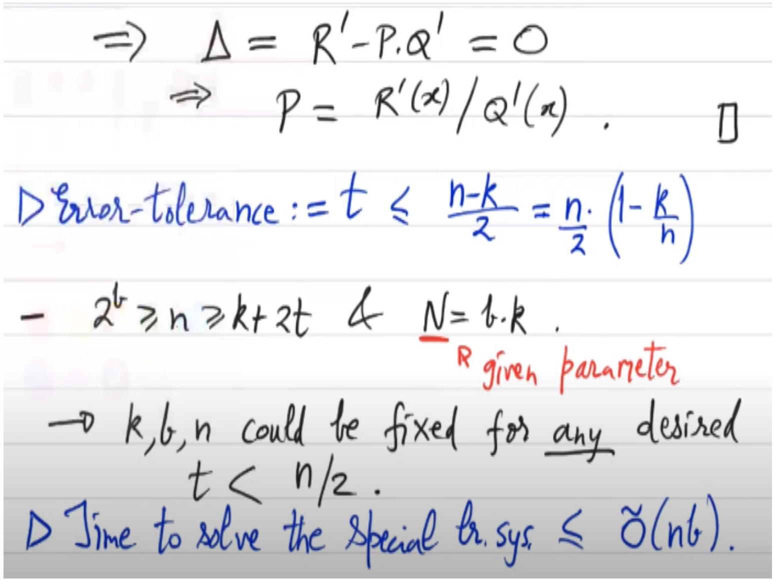
So it is not, it is a feasible system. It is not infeasible. Okay, let me change that tilde to prime. I will use prime. So let Q prime R prime be some other solution. So consider this polynomial which is R prime minus P times Q prime, okay. What can you say about this polynomial delta looking at the linear system? Sorry?

So the linear system was any equations, but the places which were correct, n - t many places can we see that delta is vanishing on those places, right? Why can we say that? Because on the yeah when j is the correct place c j prime is c j. So that is the value of P, right. R prime Q prime is an arbitrary solutions that satisfies this. So R prime is equal to P times Q prime at least on the correct places.

So delta vanishes on n – t points in e 0 to n – 1. Basically the ones that are not e i j other than the error. What do you know about the degree of delta? So degree of P is k - 1 and Q prime is right and degree of R prime is same. So the degree of delta cannot exceed k – 1 + t. And what have you assumed about k – 1 + t? Right, n – t – 1. So now do you see the proof? Right.

So delta is a polynomial whose degree is less than n – t. But it is vanishing on at least n - t points that are distinct, right. So this means that delta is actually zero because it is happening over a field which gives us both the claims.

(Refer Slide Time: 52:56)



$\Rightarrow \quad \Delta = R' - P.Q' = 0$

$\Rightarrow \quad P = R'(x)/Q'(x)$. $\square$

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

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

R given parameter

→ k, b, n could be fixed for any desired $t < n/2$.

▷ Time to solve the special lr. sys $\leq \tilde{O}(nb)$.

So you get that P is equal to R prime x divided by Q prime x. Okay, so Q prime divides R prime arbitrary solutions of the system. Moreover, the quotient is P. That is the message. Yeah, so this is what Bob will do. Any questions? How much of t can be tolerated? So t has to be less than equal to n - k by 2. This is n – k by 2. So remember that little n is the kind of clusters in the output, right.

So if your alphabet is the field alphabet, alphabet has field elements. So in that alphabet, you have little n many is the length of the code word. And roughly half of that can be corrupted, right. So it is not n by 2, it is n - k by 2. But it is quite close to half. So you can see this as n times 1. So it is close to n by 2. This fraction is multiplying 1 - k by n. Can we estimate that what is k by n?

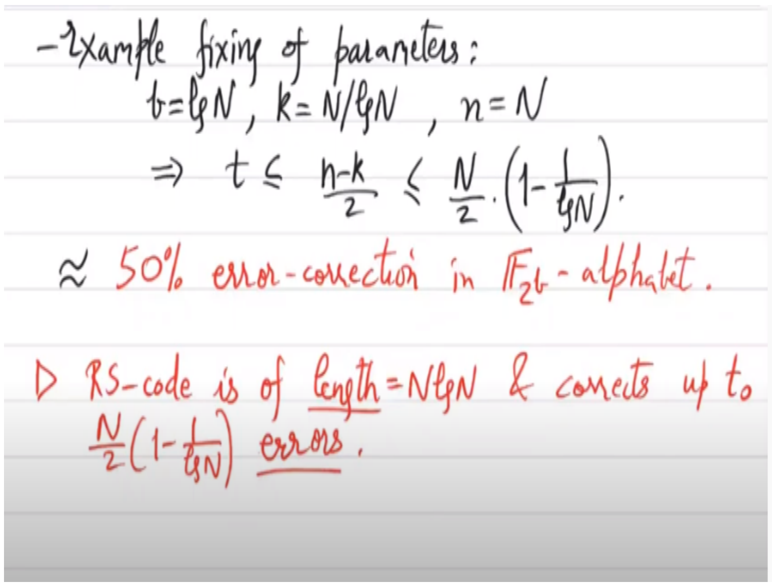
So this you wanted 2 raised to b to be at least n. And capital N was b times k. Yeah, I mean there are many possibilities. So you can take b and k to be let us say square root N, square root big N and then you can compare the two. So this key over little n will be actually quite small. So you can get arbitrarily close to little n by 2, okay. So if you want t to be 49% that you can achieve by appropriately fixing k, little n and b.

So you have started with, remember that you have started with big N that is all. That was the given parameter. And with that given you can just choose k, b and little n to get the error tolerance you need for any desired t less than n by 2. So whatever fraction you want you can do this, okay. And yeah the time complexity is clear. So this will be a very fast algorithm. You just have to solve the linear system.

So this is the only thing you have to solve. So this is this gives you essentially this little n cross little n matrix. So that you have to solve. So time to solve, trivially you can say it is polynomial in n, but even stronger claims can be made. So you can actually show that the time to solve this special system is something like O tilde nb.

So you can actually solve it quite fast and so overall we can do everything in sub quadratic. Any questions? Yeah. So let me just give a fixing. Then we can stop.

(Refer Slide Time: 59:03)



- 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.

▷ RS-code is of length $= N \lg N$ & corrects up to $\frac{N}{2}\left(1-\frac{1}{\lg N}\right)$ errors.

So example fixing is. So you can take b to be log N, k to be N over log N and little n to be N. And the t that it can handle is then, okay. So note that 2 raised to b is greater than equal to little n. And little n the second inequality is little n is greater than equal to k + 2t. So this is what you get. So you can get extremely close to n by 2. And here big N by 2. So in the finite field alphabet you are stretching N by log N to N, right.

So it is a very minor stretch. So by stretching N by log N to N and working over a finite field whose size is N as well you are able to correct errors close to 50% in the finite field alphabet, right. That is the result. So this is close to 50% correction in F 2 raised to b alphabet. So RS code is of length N sorry N log N and corrects up to N by 2. Okay, this is the final result where we can stop.

So in the finite alphabet if Alice wanted to send a message of big N many field elements, then she only has to stretch it to by a log factor instead of n square. And the errors that Bob would be able to handle on the channel is nearly half, okay. So this is a very surprising fact. It was not at all clear in the beginning a priori whether this thing could even exist, right. So this is a very fast algorithm.

This is a sub quadratic time algorithm for encoding and decoding. Any questions? Okay. Yeah, factoring was not fundamentally used here. But it is used to optimize this computation of P. So this R prime by Q prime computation, obviously, you can do it by division, but in certain applications to make it faster, they actually tried to do this by factorization.

And then it is faster in their applications. Yeah, but what was fundamentally used is finite fields, okay. So finite field is why this thing works. And it has to be a large enough finite field. Although you can see that here, this is the Berlekamp setting because the characteristic is only 2, right. So it is a very low characteristic finite field, only the smallest possible characteristic.

And then you are working in its extensions, right. So this is a good case for Berlekamp. So whenever you want to find a root or factor, Berlekamp can be applied. Okay.