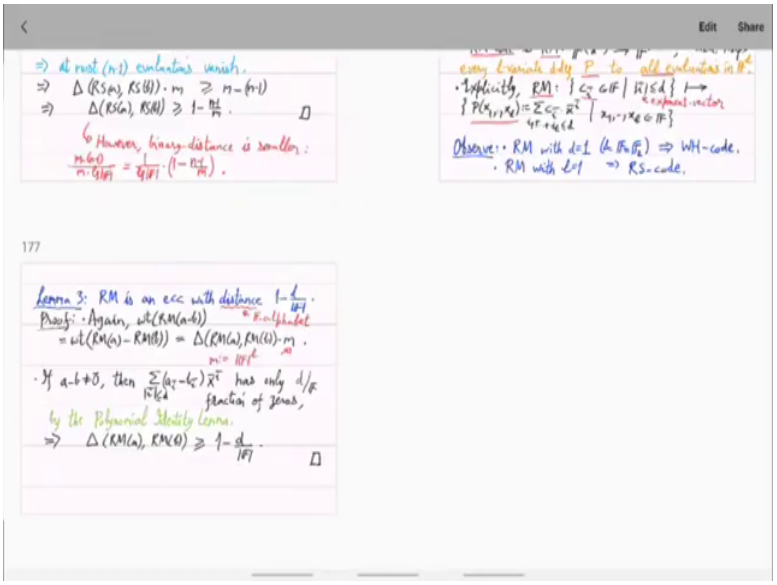
Randomized Methods in Complexity
Prof. Nitin Saxena
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

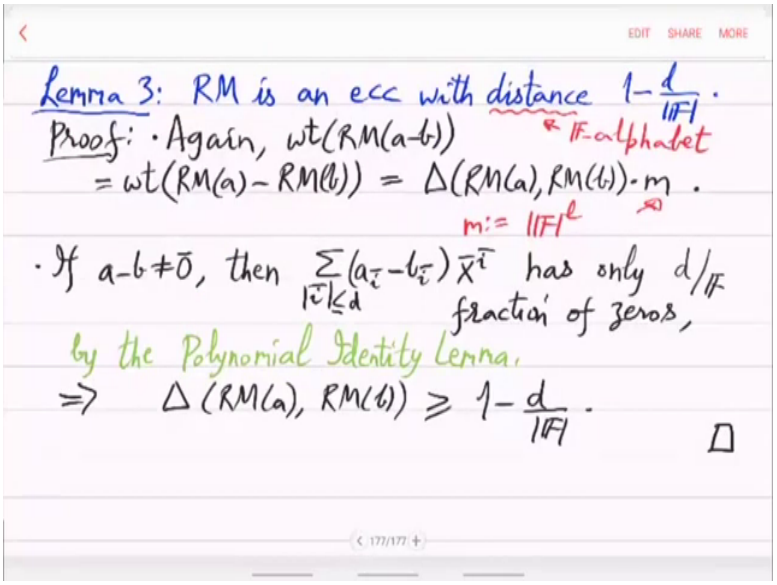
Lecture-23
Efficient Decoding of ECC

(Refer Slide Time: 00:18)



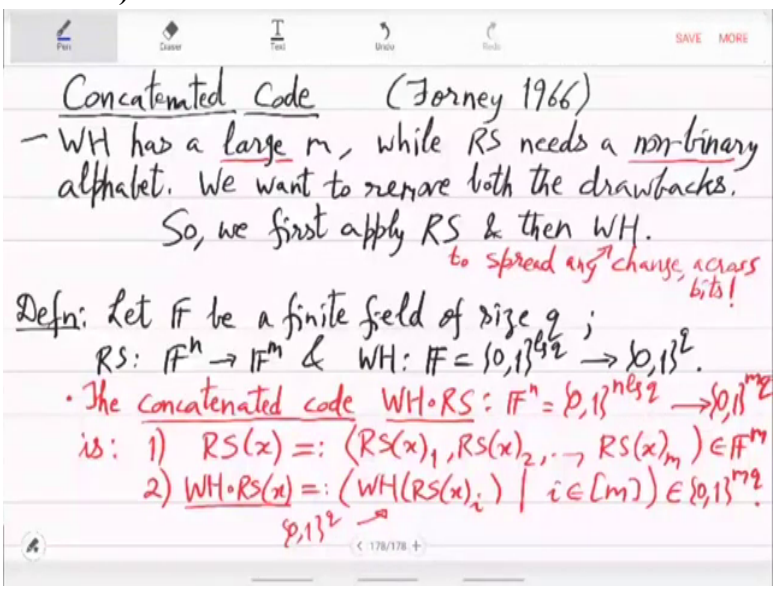
Last time we started studying 3 codes Walsh-Hadamard and we showed that the distance is half. Next, we studied Reed-Solomon code which is the most famous code we showed that the distance is in the non-binary alphabet. It is very close to 1 but remember that this is not binary if it was binary then the distance would have been half or less, but this is close to 1 because of non-binary. Then we looked at the multivariate generalization of Reed-Solomon which is called the Reed-Muller code and that has a similar distance which is 1 - d over the size of the field.

(Refer Slide Time: 01:04)



Where d is the degree of the multivariate polynomial. Next we will use both of these Reed-Solomon and Walsh-Hadamard and give a code that has the advantages of Reed-Solomon, but it is binary.

(Refer Slide Time: 01:23)



These are called concatenated codes and these are also old. It was given by Forney in 1966. So, Walsh-Hadamard is a very good distance wise it is also binary, but the problem is it is very long. So, Walsh-Hadamard has a large length m while Reed-Solomon is a non-binary alphabet, Reed-Solomon you cannot do for binary alphabet because it needs a field and the field has to be big enough.

So, Walsh-Hadamard has the problem of large m Reed-Solomon requires non-binary alphabet we want to solve both problems the solution will be via concatenation. Put the drawbacks

disappear you want to remove both the drawbacks? So, what we do is we first apply Reed-Solomon and then we apply on the non-binary alphabet these field elements we apply the Walsh-Hadamard code.

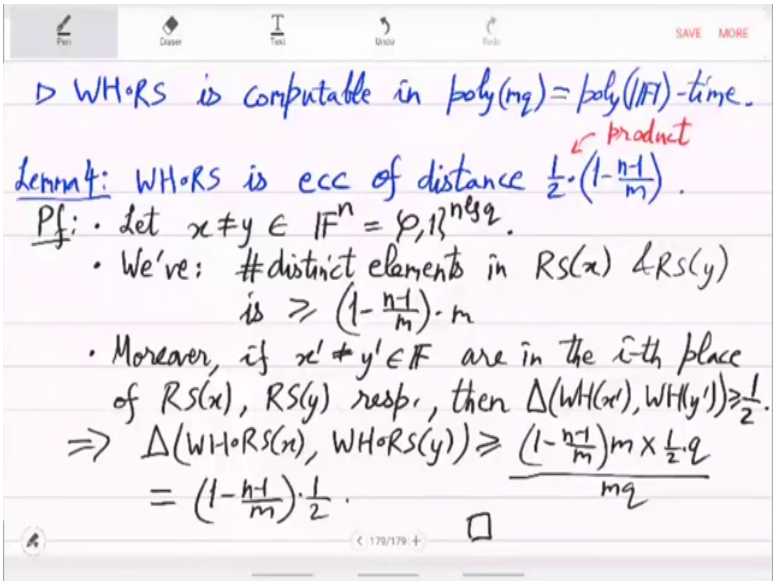
We need to apply Walsh-Hadamard on these field elements so, that any change in the field element should spread around the bits. So, let us define this formally. Reed-Solomon will spread it across field elements and then Walsh-Hadamard will spread it across the bits. So, let F be a finite field, finite alphabet of size q Reed-Solomon is the map on n length message in this F alphabet it stretches it to F and Walsh-Hadamard this will stretch field element viewed as binary.

So, $\{0, 1\}^{\log q}$. Actually what this is let me first write F_q. So, view F as binary space log q bits and then Walsh-Hadamard you know is exponential. So, it will stretch log q to q. The concatenated code Walsh-Hadamard, Reed-Solomon, this is the map which views F_q^n as $\{0, 1\}^{n\log q}$. So, first this F^n is mapped to F^M and then F is mapped to a longer string which is q long.

So, you get qm and how is it mapping? First maybe I should write it like this in the first step what Reed-Solomon will do is it will give you the first field element, the second field element and the m^{th}. Basically, on a string x which is an N tuple of field elements, it will stretch to M field elements. So, this is an F^M and in the second step it will apply Walsh-Hadamard on each of these field elements.

So, the i^{th} symbol view, it as a Boolean string and then while shadow mode will stretch it further. You will get each symbol will be mapped to $\{0\ 1\}^q$ and the whole thing is this whole thing is then in $\{0,1\}^{mq}$. This is the definition this is how Walsh-Hadamard, Reed-Solomon concatenated code acts. You know that Reed-Solomon is an efficient code efficient in the sense of poly m log q and Walsh-Hadamard will take time poly q. So, overall, this is polynomial in m and q and m is also comparable to field size.

(Refer Slide Time: 08:55)



You can observe that Walsh-Hadamard Reed-Solomon is computable in poly m q which is equal to poly field size q is completely deterministic polynomial-time computable. Now, what is the distance? Last time we did lemma 3 which was the distance for Reed-Muller. Let us now repeat a similar analysis for concatenated code. Let us call it lemma 4. So, Walsh-Hadamard, Reed-Solomon is an error-correcting code of distance.

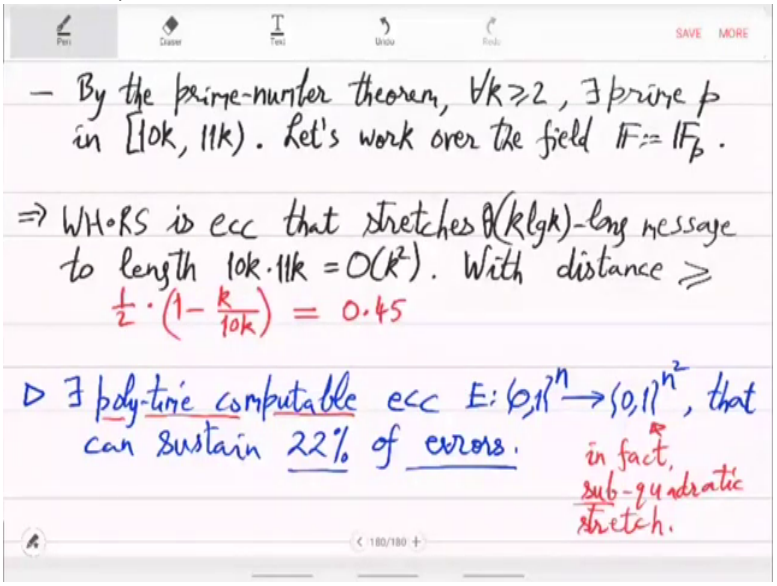
So, the Reed-Solomon code has distanced 1 - (n -1) / m. Two different strings or 2 different n tuples from the field will be mapped to tuples which are m tuples and they differ in these many places in those places Walsh-Hadamard will when you look at the binary representation Walsh-Hadamard will differ in half of the places. It is actually a product, this is the distance its product of the distance. The proof is quite straightforward.

Let x and y be basically n tuples from the field which you are now viewing as a binary string. We know that number of distinct elements in Reed-Solomon encoding of x and Reed-Solomon encoding of y is this number is 1 - (n - 1) / m times m distance times the stretch length the codeword length in these many places are RS(x) and RS(y) differ. Moreover, if x', y' field elements are in the i^{th} place of RS(x), RS(y) respectively.

Then in the Walsh-Hadamard stretch or encoding the distance is half. This means that overall if you look at the binary distance between the concatenated code of for x and that for y this is the places where you get the difference in the Reed-Solomon level is this and the places amongst these now when you apply Walsh-Hadamard the differences is in half times q places and the total length, in the end, is mq.

So, you get 1 - (n - 1) / m times half as claimed. So, this is now you have a binary code with a decent distance. If you take the field to be let us say 10 times n. So, you stretch 10 times then this distance is very close to half. Let us do that analysis. What is it that we are getting in the end? The concatenated code relates to some parameter chasing.

(Refer Slide Time: 14:52)



By the prime number theorem. Why the prime number theorem? Well, because you want to work in a finite field. So, you can work with a prime field and for that, you need the distribution of primes how so, how many prime fields are there actually, so, we know by the prime number theorem that there are a lot of primes, in fact, no matter what k you pick, there exists a prime p in the interval 10k to 11k there is a there is always a prime between 10k and 11k.

Let us work over the field this prime field, integers mod p. This is kind of setting your m your M is around 10k and I will think of k as n because the field is 10 times n. Now, let us go through the concatenated code analysis or parameters specific to this prime. So, the concatenated code was Walsh-Hadamard on Reed-Solomon is an array error-correcting code that stretches k log k long message.

Let me not get into the exact constants let me keep this part approximate the stretches around k log k log message to length mq in the previous notation m here is kind of p So, I take this 10k and log k will be stretched to k log k will be stretched to p and let us not be too precise. To say this is length around 10k times 11k. That is the upper bound k order k^2 basically.

So, by the previous analysis of the concatenated code k log k is being stretched to around k^2 and the finite field that we have picked is this prime field p is a prime number between 10k to 11k and the distances. What is the distance parameter? That is half times 1 - n / m. So, n is we are taking it to be k and n we are taking to be the prime p which is 10k. This means 0.45 like so, distances 45%.

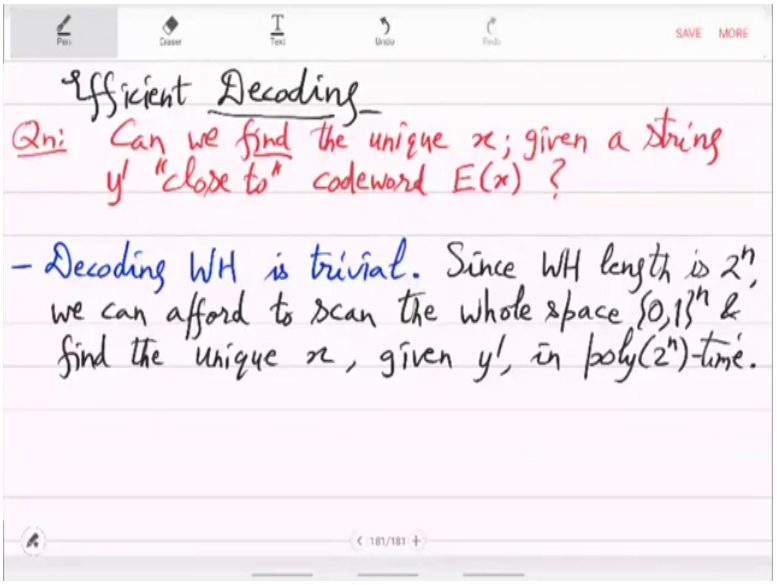
The error that this concatenated code can and it is a binary code that it can handle is half of this, so, which is more than 22%. Let us note that down that is our important observation. It is important especially in practical applications so for all numbers n there exists a polytime computable error-correcting code E which will stretch n to n^2 that can sustain pretty 22% of errors.

So, there is an error-correcting code which is polytime computable in the message length it stretches to around quadratic. In fact, it stretches to less than quadratic because it is stretching k log k to k^2, it is less than quadratic, it is subquadratic stretch and most importantly the errors that it can handle is 22%. So, by handle, we mean that even if there are 22% errors in the channels of communication, the corrupted codeword will have a unique message string.

They will be a unique code word that will be close to a corrupted codeword. So, this is an amazing property when we started this problem, we had no idea that such a thing exists? Because the key thing here is polytime computable. It is a really efficient error correcting code and it is able to handle at least theoretically a large fraction of errors under sub quadratic stretch.

The question now is all this is good theoretically, but to be actually useful, there should be a decoding algorithm. Right now, the decoding algorithm seems to be just going over all the code words and finding the one which is closest to your corrupted codeword. But that clearly is very inefficient the space is just too huge to search for in practice. So, we need a fast decoder as well.

(Refer Slide Time: 22:48)

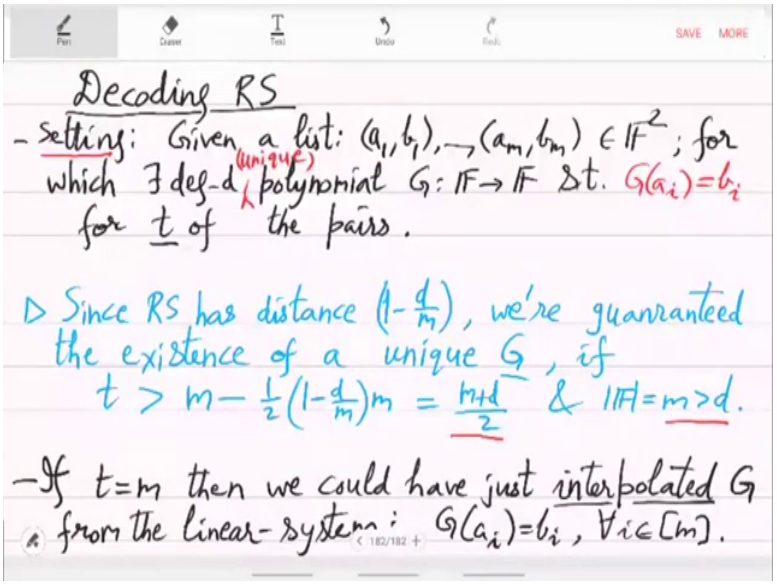


How do you do efficient decoding that is the question. So, can we find the unique x given corrupted code word or given a string y' close to the code word E(x). So, you want to find the unique x such that the codeword corresponding to E(x) is our error-correcting code when you send over the channel it will get corrupted to y' but it will not be too corrupted it will be close to in terms of hamming distance it will be close to E(x).

Given y' can you find x? That is the question so, decoding Walsh-Hadamard is trivial. Because Walsh-Hadamard is already such a long code, to reach to in length, that in that much time you can go over the whole space and find x. Since the length is 2^n already we can afford to do scan the whole space $\{0, 1\}^n$ and find the unique x given y' in poly 2^n time.

This is a simple algorithm. Search the whole space this is what we call efficient or fast because the length was already exponential rate. So, in terms of that, this is polynomial time or a more interesting challenge will be the question how to decode Reed Solomon and then read Mueller and finally, the concatenated code.

(Refer Slide Time: 26:06)



So, decoding Reed-Solomon: Let us fix the question clearly what are we given and what do we want to find? So, you are given a list of evaluations m evaluations of a polynomial is what you are given. So, a_1, b_1 to a_m, b_m you are given for which there exists a degree d polynomial G such that G is an evaluation at a_i is b_i and remember that G is unique as well. Let me write that here for which there exists a unique polynomial such that $G(a_i)=b_i$ for t of the pairs.

It is not that $G(a_i)$ is equal to b_i for all the pairs; for all a_i that is that would be the case when there is no corruption that happened to the code word which was sent over the channel. So, obviously, in reality, there will be corruption maybe 22% of the pairs are corrupted. It is only the remaining 80% or 78% of the i's for which $G(a_i) = b_i$ and from this you want to compute or find G. Now since RS has Reed-Solomon has distance 1 – d / m rate remembers G is degree d.

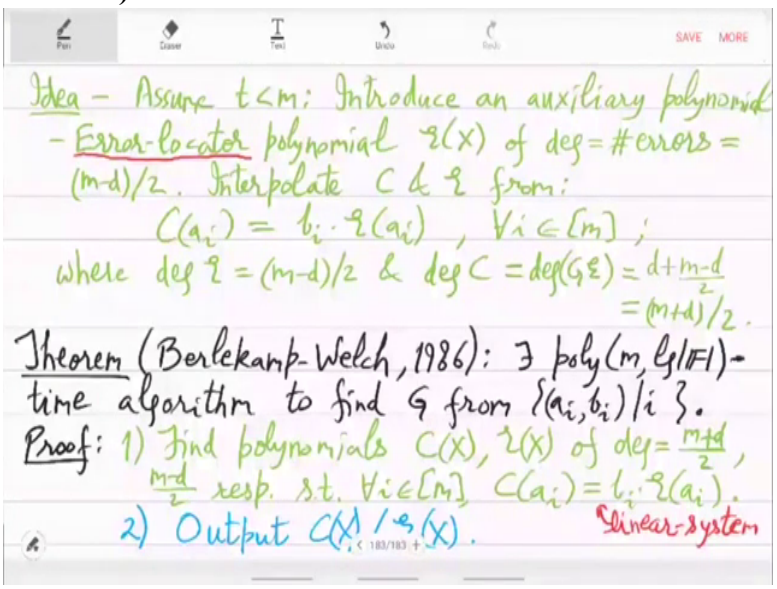
Since Reed-Solomon has distanced this much we are guaranteed the existence of a unique G if t is big enough, t should be m minus the errors that you can tolerate which is half of that d / m times m, which is m + d / 2 and the field size which is m, this should be bigger than the degree. It is an interpolation question. Obviously, you should have enough field elements otherwise you cannot interpolate a d degree polynomial.

So, remember these two parameters, the t is more than n + d / 2 and m is more than d. In particular t is more than d. That sanity check is passed I mean, you should have more than d pairs where evaluation of G is correct. Otherwise, you cannot hope; it is impossible to get a degree d polynomial G. But we want to give an algorithm to find G and the problem we are

facing is that we do not know which i's are correct, which i's are bad. So, the number of subsets of M is too large we cannot go over all possibilities.

How do we solve this problem algorithmically? Well, if t was m that is everything is correct, then we can just use interpolation could have just interpolated G from the linear system $G(a_i) = b_i$ for all i in 1 to m. So, if there is no error, then this is just an interpolation problem, you can solve it, you can find G. But what if t is less than m, case so, we will do that now. Assume that t is smaller than m which means that m - t pairs are erroneous and you do not know which pairs are good, or which pairs are bad. Still, you want to find G.

(Refer Slide Time: 32:57)



The idea is in this case, to compensate for the errors, you introduce an error locator polynomial, which is called the error locator polynomial. We will call it E(x) which has degree equal to the number of errors and the number of errors you know can be m - d / 2. Now, obviously, we do not know which pairs are bad which b_i's are good or b_i bad. So, we will also not know the details of this error locator polynomial.

But we can set up an equation so, interpolate or find C and E from the following equation $C(a_i) = b_i E(a_i)$. Now, since we do not know which i's are good or which are bad, we have to treat all of them equally. Set up this equation C(i) = b(i) E(i) for all i this gives you across all i's, you have now m constraints where degree of E is already set to m - d / 2 and degree of C is C is supposed to mimic G. Now C will mimic something like G times E. So, this will be degree of G times E.

That degree will be d + m – d / 2 which is equal to m + d / 2. m + d / 2 has another

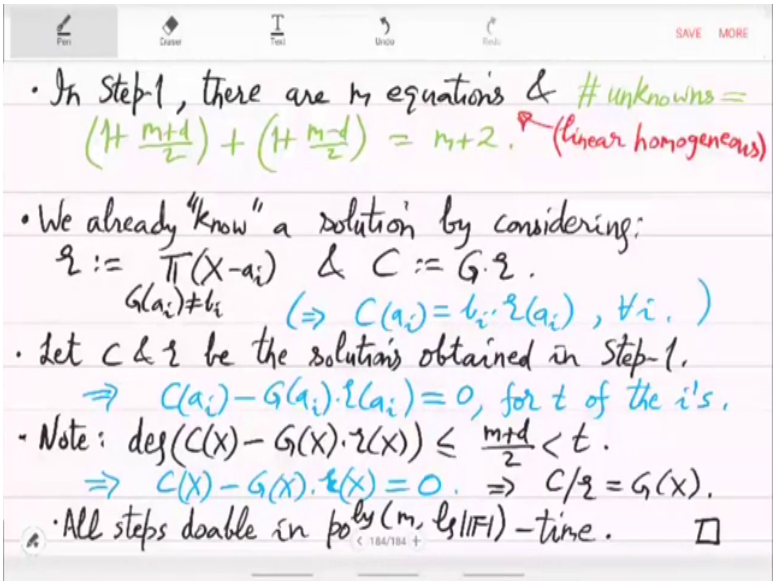
importance you know that t is bigger than this. Now, you can see that the number of places where a i , b i are correct is more than the degree of this system. There is some hope and so, when you solve this linear system, you will get very useful information. That is how people came up with Reed-Solomon decoding.

Let us make it a theorem. This is due to Berlekamp and Welch they showed that it exists polynomial time which is polynomial in m and log of the field size. Field size you always assume bigger than m which is bigger than n, but this is a truly deterministic polynomial time algorithm. There exists polytime algorithm to find G from a_i, b_i. So, whatever this data is with the guarantee that t of them are correct t of the pairs are correct, you can find G. So, the algorithm is as follows.

So, the first step is to find polynomials C(x), E(x) have degrees m + d / 2, m - d / 2 respectively such that for all i, $C(a_i) = b_i E(a_i)$ note that this is a linear system. From the linear system, you can find a solution. You just need to use a linear system solver and that will give you polynomial C(x), E(x) and then you just output, the understanding was that C is G times E C we just output the ratio output C(x) / E(x).

This is it is a simple algorithm it is certainly efficient, but it is not at all clear why C / E will be a polynomial and why would this be the unique G let the algorithm tells nothing about gives you no hints. So, let us try to prove that. Let everything is in the proof actually. Based on the idea that we discussed, we are thinking of E as is the error locator So, it will basically intention is to collect these bad a i's in the error locator.

(Refer Slide Time: 40:23)



In step 1 there are m equations and how many unknowns, so the number of unknowns is 1 + m + d / 2 which is for C and 1 + m - d / 2 that is for E, which is m + 2 there are m equations m + 2 unknowns and it is a homogeneous linear system. These are linear homogeneous. Since it is linear homogeneous with many unknowns, it will always have a solution and in fact.

You know, one kind of solution we already know a solution by considering take E to be the bad a_i's and C to be G times E and you can see that $C(a_i) = b_iE(a_i)$ for all i, because for the good a_i is $C(a_i)$ is always G times a_i and you know that $G(a_i) = b_i$ for good a_i's and for the bad ones, this a_i will be 0. Either a_i is 0 or when it is non-zero then still, because of $G(a_i)$ or equal to b_i this equation is satisfied. This equation is always satisfied and that is the beauty of this idea.

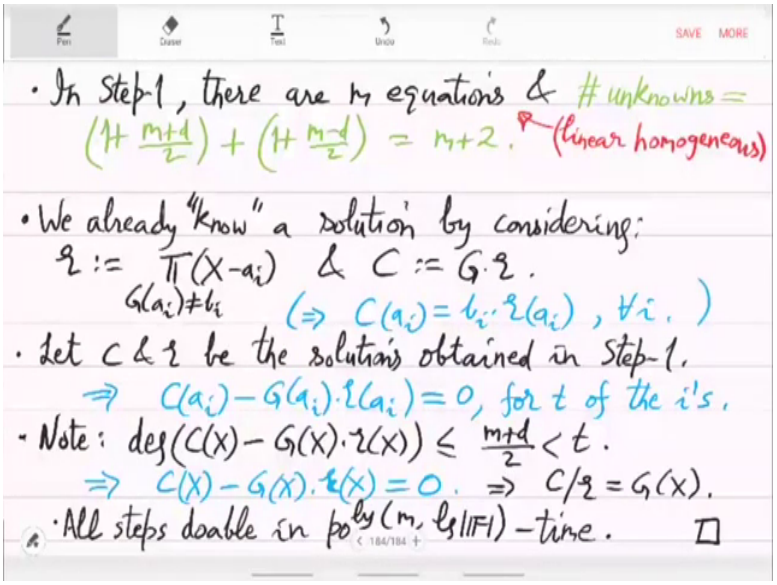
So, we already know a theoretical solution, it is a linear system, we can solve it. So, we will get some solution maybe this one or something else. So, let C and r be the solutions obtained. in step 1. The linear systems solver gives you some C and r it might not be this ideal case C and E. How do you know about C / E, it may not be G. Let us see that. So, you know that for many values for the good a_i's $C(a_i) - G(a_i)E(a_i) = 0$ for t of the i's, paid this you know because for t of the i's $G = b_i$.

You know that whatever the solution, C and r, they will satisfy this, at least for t of the i's and for all of them, they satisfy $C(i) = b_iE(a_i)$. Now, note that if you look at the degree of C(x) – G(x)E(x) this degree is well C(x) is degree m + d / 2 and G E is also the same. It is at most m + d / 2 and we have taken t to be larger than that? What t did we take? Greater than m + d / 2.

So, degree of C - d times C is smaller than t. But the above statement in blue says that this polynomial vanishes for t of the i's, which means that this polynomial is 0 and that, so, this is the proof this finishes the proof. This is the reason why no matter what C and E you get in step 1 C / E is always G and that is what you output in step 2. So, all steps are doable in polynomial in m which is basically the degrees involved and log of the finite field size you can do addition, multiplication and also division.

But all these field operations during operations can be done in polynomial time. So, this finishes the decoder of Berlekamp and Welch for the Reed-Solomon code k this is one of the nicest algorithms in algebra and this is what we will build on as we move forward to more complicated encodings.

(Refer Slide Time: 47:26)



Let us now immediately apply this to decoding concatenated code Walsh-Hadamard on Reed-Solomon. For Walsh-Hadamard, Reed-Solomon code which is basically this map $\{0,1\}^{n \log q}$ to $\{0,1\}^{mq}$ bits there exists a polynomial in the finite field. So, this is the size of the field is an upper bound on both n and m there exist of polynomial in q time decoder if the fraction of errors if the other fraction is less than half of the distance which is it has to be actually half for each of these.

For Reed-Solomon you have to half which will give you half - n -1 / 2m and then on top of this Walsh-Hadamard also you have to have which will be half of half it is 1/4, so, it is kind of 1/4 of Reed-Solomon. For our parameters when Reed-Solomon was handling 22%. This

will handle 5% slightly more than 5%. So, let y' be close to y which is a concatenated code word.

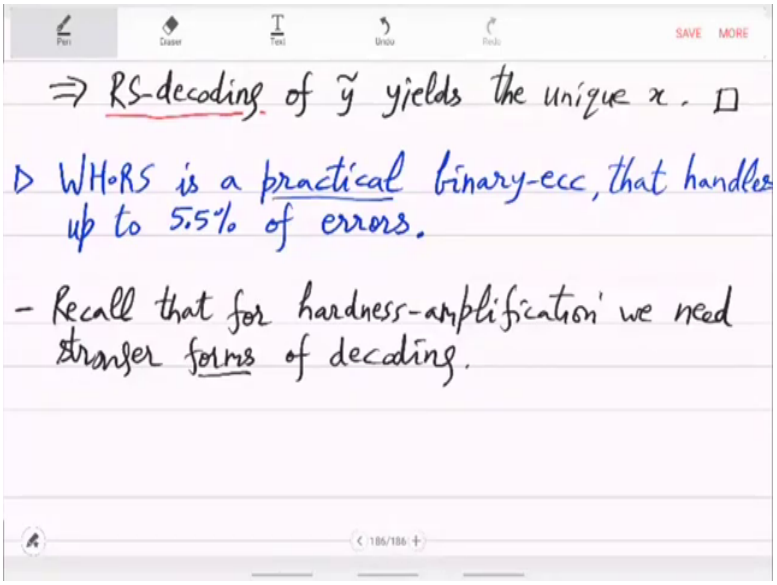
That is Walsh-Hadamard on Reed-Solomon symbol ith symbol for all i and y' distance to y is as given above the error fraction in that sense it is close. So, there is some corruption let us say up to 5% corruption now, the hypothesis on this error fraction implies that number of symbols are ith symbol i such that Walsh-Hadamard on the symbol has a lot of errors at least q / 4 errors number of such i's cannot be large.

The reason is that if it is large and Walsh-Hadamard in each Walsh-Hadamard the symbol the errors are also large then overall it will exceed the bound that the hypothesis has put. By averaging basically you are getting that the number of symbols on which Walsh-Hadamard is making a lot of mistakes is smaller than half - n -1 / 2 m times m. That is giving you m - n + 1 / 2. The Walsh-Hadamard encoding of at most these many symbols has a lot of errors in the code in those symbols.

The remaining case Walsh-Hadamard of the symbol has fewer than q / 4 for errors and that is where you then do Walsh-Hadamard decoding. Walsh-Hadamard decoding will yield m tuple which we call $\tilde{y}$ where basically more than half of the cases $\tilde{y}_i$ is correct. So, locally you have decoded using the Walsh-Hadamard WH decoder and more than half of these are correct values.

You will not know this, but it is a guarantee that WH decoder gives you. You would not know which i's are good, but you will know the number of i's that are good. So, with $\tilde{y}_i$ equal to correct by i^{th} symbol for greater than m - m - n + 1 / 2 which is m + n - 1 / 2 of the i's. You do not know which ones but you know the number of i's. It is more than m + n - 1 / 2. After this WH decoder now you have symbols where lot of the symbols are correct you do not know where they are located but the guarantee is many of them are correct and at this point you should invoke the RS decoder that will give you x.

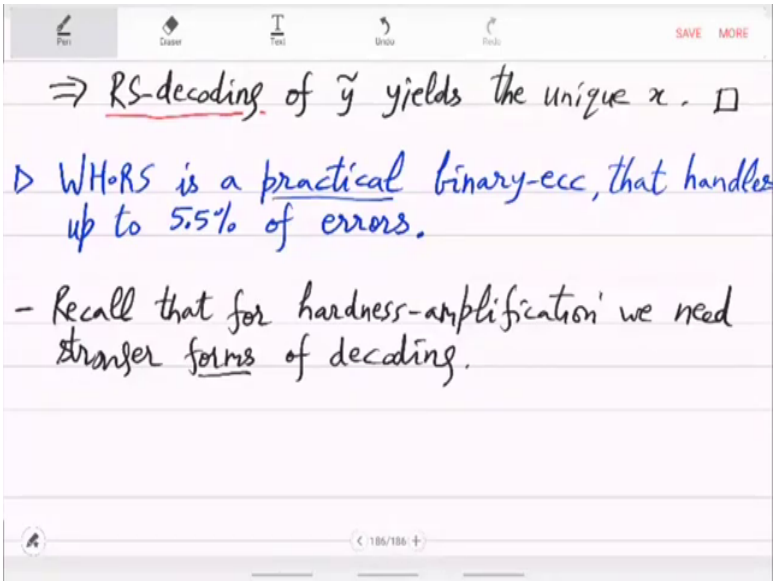
(Refer Slide Time: 54:51)



So, RS decoding of $\tilde{y}$ is the unique x because RS decoder needs how many correct i's more than m plus degree by 2 and in this case degrees n - 1 say it is more than n + d / 2 by the previous RS coder. So, what you have learned is Walsh-Hadamard, Reed-Solomon is a practical binary ECC. It is both binary and a linear error correcting code that encoding decoding both are polynomial time. For Reed-Solomon what did you learn before that it can handle 22% of errors.

Let us just do 1/4 of that, so 5.5% of errors. So, if there are 5% of errors in the channel then you can use Walsh-Hadamard, Reed Solomon coding. All these are beautiful results both theoretically and practically they are very nice and in a way unexpected also. Now, remember our goal was to do amplification of function hardness. So, for that, as we alluded before, not just decoding, but stronger forms of decoding will be required. So, the first form that I will define stronger form of decoding is local decoding. Recall that for hardness amplification, we need stronger forms of decoding.

(Refer Slide Time: 58:02)



The next type of decoding concept is local decoding. We will define the concept, it is basically that you will be allowed only to query the code word or the corrupted codeword string, a few places and from those few places that when you query the bits, you have to guess let us see the first bit of x because so the goal is not to find the whole x, but the first bit of x and in the codeword or corrupted code word, you are allowed only a few locations to query. We will define it and then we will actually do it for these 3 or 4 coding mechanisms that we have seen in the next class.