


**Secure Computation: Part II**  
**Prof. Ashish Choudhury**  
**Department of Computer Science and Engineering**  
**Indian Institute of Science, Bengaluru**

**Lecture – 19**  
**Reed-Solomon Error-Correcting Codes**

(Refer Slide Time: 00:24)

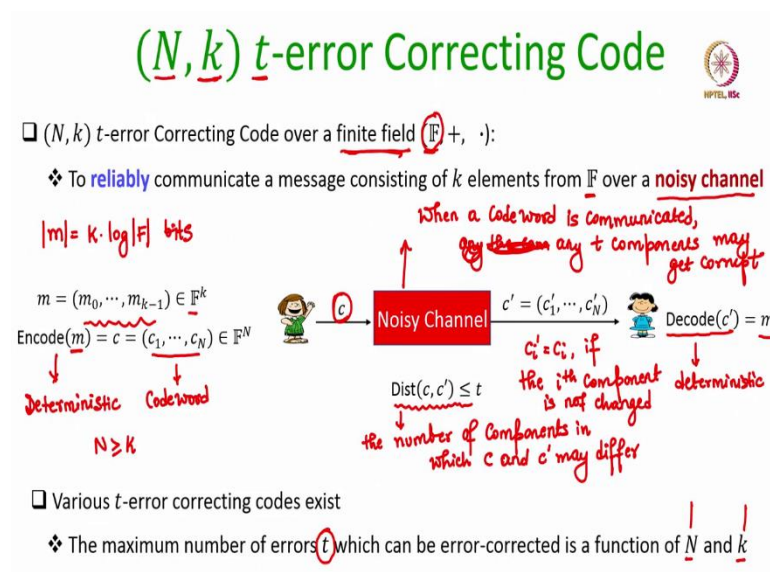
## Lecture Outline



- ❑ Error-correcting codes
  - ❖ Reed-Solomon (RS) error-correcting codes

Hello everyone. Welcome to this lecture. In today's lecture, we will discuss about error correcting codes. Specifically, we will discuss about Reed-Solomon Error Correcting Codes which are also known as RS codes, after the name of Reed and Solomon; attributed to Reed and Solomon who invented this code. We will be using these error correcting codes extensively later during the discussion, during the design of our MPC protocols.

(Refer Slide Time: 00:51)



So, let us first start discussing about  $(N, k)$   $t$ -error correcting codes, what exactly is their purpose, how they operate, the syntax, the semantics and the underlying properties. So, we will be considering error correcting codes, where all the operations will be done over some finite field. And remember in the last lecture we have seen what exactly a finite field is.

So, it is a set of elements with two operations,  $+$  and  $\cdot$  which satisfy certain axioms. Now, we will be working with a finite field. And there are plenty of candidates for instantiating a finite field. Now, what exactly is the goal of an  $(N, k)$   $t$ -error correcting code? The goal is basically to reliably communicate a message over a noisy channel, where the size of the message is  $k$  namely. It consists of  $k$  elements,  $k$  symbols from the field  $\mathbb{F}$ .

So, you can imagine that we have a sender who has a message consisting of  $k$  elements from the set  $\mathbb{F}$ . So, you can imagine that the sender's message is basically  $k \cdot \log |\mathbb{F}|$ , because each element from the field can be represented as  $\log |\mathbb{F}|$  number of bits. So, if sender has a message which is a binary string consisting of  $k$  times log of bits; it can be abstracted, it can be modelled as if the sender has  $k$  number of elements from the field.

The reason we are considering the bits as elements of a field is that we will be designing our error correcting code, where all the operations will be done, over the field. Now, this error correcting code will have two operations: an encoding operation and a decoding operation. So, the encode operation is a deterministic algorithm which takes as input the message  $m$  and it produces an output which we call as code word.

This consists of  $N$  elements from the field, where  $N$  is definitely at least as large as the message. Namely, the number of components in the code word is at least the same as the number of components in the message. Now, there is a noisy channel through which the sender is connected to a receiver. And this noisy channel has the property that when a code word is communicated any  $t$  components may get corrupt.

Which  $t$  components? Their exact locations will not be known whether it is the first  $t$ , whether it is the last  $t$ , whether it is every alternate component. No, it would not be known, but it will be known that up to  $t$  components may get corrupt whenever the code word  $c$  is communicated over this noisy channel.

So, that is denoted by saying that the distance between the sent code word and a received code word is at most  $t$ . And distance means the number of locations or the number of components in which  $c$  and  $c'$  may differ. So, let us denote the code word received by the receiver as  $c'$ , whose components are  $c_1', c_2', c_N'$  and  $c_i'$  will be same as  $c_i$ , if the  $i$ th component is not changed.

Now, the receiver's goal is to somehow recover back the sender's message by applying a decoding operation which is also going to be a deterministic algorithm. So that, even if up to  $t$  components in the received vector are corrupt, somehow the receiver can recover back the message; that is the goal of an  $(N, k)$   $t$ -error correcting code. And there are plenty of real-world applications for this error correcting codes. Say for instance whenever communication is coming from satellite to the stations, base stations on the earth.

Then it could be possible that when the signals are sent, some errors are introduced due to the noise in the channel. There the goal will be then to somehow introduce a redundancy while communicating the signal through this encoding operation so that, even if some noise is introduced, the receiver can recover back the original signals. Or, whenever we are talking over the mobile phone and some communication gets disturbed due to the noise which is introduced.

The error correcting codes can help us to ensure that even if some noise is introduced, the communication between the two parties, the two ends is error free, it's smooth. So, we have three parameters here. The size of the message, that is  $k$ , the length of the code word

$N$  and how many errors we would like to tolerate and there are various  $t$ -error correcting codes which exist.

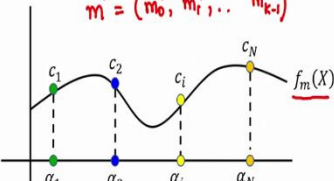
The maximum number of errors  $t$  which can be tolerated which can be error corrected is a function of your message size and the code word size. So, there are some well-known bounds which determines how many errors we can tolerate for a given  $k$  and a given  $N$ . And we cannot beat those bounds. So, we will not go into those bounds and those details, because this is not a course on error correcting codes.

(Refer Slide Time: 08:48)

## Reed-Solomon Code: Encoding Algorithm

□ Reed-Solomon (RS) **encoding** algorithm:

- ❖ Input:  $m = (m_0, \dots, m_{k-1}) \in \mathbb{F}^k$  (k-1) degree polynomial
  - $f_m(X) \stackrel{\text{def}}{=} m_0 + m_1 \cdot X + \dots + m_{k-1} \cdot X^{k-1}$  --- **message-encoding polynomial**
- ❖ Output:  $c = (c_1, \dots, c_N) \in \mathbb{F}^N$  --- **RS codeword** || A collection of N distinct values of the polynomial  $f_m(x)$ 
  - $c_1 = f_m(\alpha_1) \quad c_2 = f_m(\alpha_2) \quad \dots \quad c_i = f_m(\alpha_i) \quad \dots \quad c_N = f_m(\alpha_N)$
  - $m' = (m'_0, m'_1, \dots, m'_{k-1})$   $f_{m'}(x) = m'_0 + m'_1 x + \dots + m'_{k-1} x^{k-1}$
  - $c'_i = f_{m'}(\alpha_i)$
  - (α<sub>1</sub>, ..., α<sub>N</sub>): publicly-known, distinct elements from  $\mathbb{F}$



We will stick to a specific instance of error correcting codes which will be useful later for us and this is the Reed-Solomon code. So, let us try to understand the encoding and the decoding operations for the Reed-Solomon codes. So, the encoding algorithm is very simple, you have the message consisting of  $k$  elements from the field. To generate the code word, we first construct the message encoding polynomial.

I denote it as  $f_m$  and it is a polynomial of degree  $k - 1$ . So, it is a  $k - 1$  degree polynomial and its coefficients are the elements of the message vector. What will be the code word? The code word will be consisting of  $N$  elements from the field,  $c_1$  to  $c_N$ . And what are these elements?  $c_1$  is the evaluation of the message encoding polynomial at  $X = \alpha_1$ ,  $c_2$  is the value of the message encoding polynomial at  $X = \alpha_2$ ,  $c_i$  is the value of the message encoding polynomial at  $X = \alpha_i$ .

And  $c_N$  is the value of the message encoding polynomial at  $X = \alpha_N$ . Now what are  $\alpha_1, \dots, \alpha_N$ ? They are publicly known system parameters which are distinct elements from your field. So, you can imagine that Reed-Solomon code word is nothing but a collection of  $N$  distinct values of the polynomial  $f_m(X)$ , namely the message encoding polynomial evaluated at  $N$  distinct values.

So, you can imagine that the code word components are basically  $N$  distinct points on this polynomial  $f_m(X)$ . I stressed that the components  $\alpha_1, \dots, \alpha_N$ , they can be any  $N$  elements from the field. They will be publicly known, and they are fixed once for all while computing the Reed-Solomon code word for any message. You have, if you have another message say  $m'$  whose message blocks are  $m'_0, m'_1$  like that  $m'_{k-1}$ .

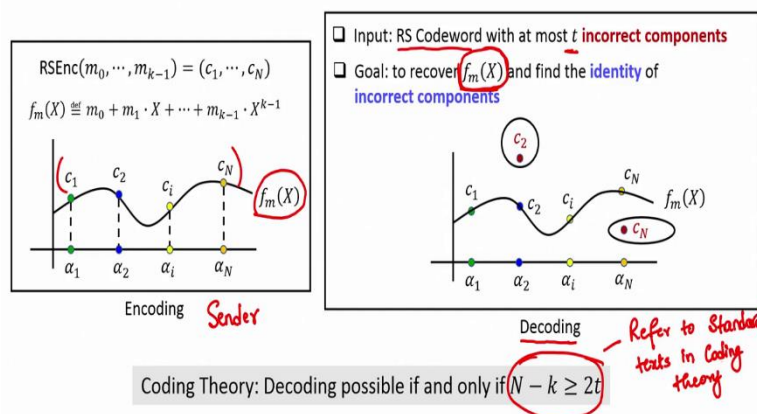
Then, we will construct a message encoding polynomial for  $m'$  whose coefficients will be  $m'_0, m'_1, \dots, m'_{k-1}$ . Its degree will be  $k - 1$  and now let us denote the code word components as  $c'_1, c'_2, \dots, c'_N$ . So,  $c'_1$  will be the value of this message encoding polynomial at  $\alpha_1$  and so on. So, the evaluation points namely the points at which the polynomials are going to be evaluated, they do not get changed for different messages, they remain the same.

It is only the message encoding polynomial which will get changed if the message gets changed. So, that is the simple Reed-Solomon encoding algorithm. You simply form a polynomial out of your message by treating the elements of the message vector as the coefficients of your polynomial. And now you evaluate the polynomial at  $N$  distinct values that constitutes your Reed-Solomon code word.

(Refer Slide Time: 13:09)

## Reed-Solomon Code: Decoding Algorithm

$\alpha_1, \dots, \alpha_N$ : publicly-known, **distinct elements** from  $\mathbb{F}$



Now, let us try to understand how the decoding algorithm works. So, you have the  $N$  publicly known distinct elements from the field. And, in the box I have shown the encoding algorithm. Now for the decoding algorithm, the input will be a vector of length  $N$ . Ideally, it should correspond to some Reed-Solomon code word, but that is need not be the case because at most  $t$  components of the input for this decoding algorithm may be incorrect.

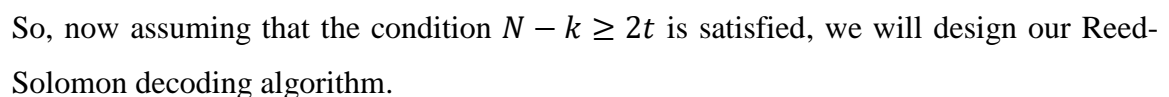
So, suppose say for instance sender has computed this polynomial and this was the code word  $c_1$  to  $c_N$ . But, when it reached the receiver, suppose  $c_2$  got changed and it is now a different  $c_2$ . And, say  $c_N$  got changed and now it is a different  $c_N$ . Receiver will not be knowing which component is the correct component, which component are the wrong component. But it will be knowing that there could be up to  $t$  incorrect components in the vector which it has received.

The goal of this decoding algorithm is to somehow recover the message encoding polynomial and find the locations, where the corresponding components are the incorrect ones; that is the goal of this decoding algorithm. So, in some sense you can imagine that this decoding algorithm is nothing, but a way of recovering the original polynomial from a subset of good points and a subset of bad points, where the exact identity of the good points and bad points is not known.

But we only know the ratio the number of good points and the number of bad points which receiver might have. Now, first thing before we try to even attempt to design a decoding

Namely, the difference of the difference between the size of the code word and the message should be at least twice the number of errors which you want to error correct. This condition is not there, then you can never design a decoding algorithm which will help you to get back uniquely the message encoding polynomial; that means, the recovery may be now ambiguous. So, we will design the decoding algorithm assuming this condition is satisfied.

(Refer Slide Time: 16:46)



So, what is our goal? Our goal is to get back the unknown  $k - 1$  degree polynomial. Why it is unknown? Because, we are now considering from receiver's perspective, receiver has received a code word of size  $N$ . Some of the components are correct, some of the

components have been changed. It does not know which components are the right one, which components are the wrong one.

But it knows that those components would have been generated from some message encoding polynomial whose degree would have been  $k - 1$  because the parameters  $(N, k)$  and  $t$  are publicly known. So,  $(N, k)$  and  $t$  are public parameters here. What the receiver knows? Well receiver has received a vector of size  $N$ . So, say it has received the vector  $c_1, c_2', c_3, c_i$  and  $c'_N$ ; that means, we are considering the case when the second and the  $n$ th component are corrupted. And it also knows the points evaluation points  $\alpha_1, \dots, \alpha_N$

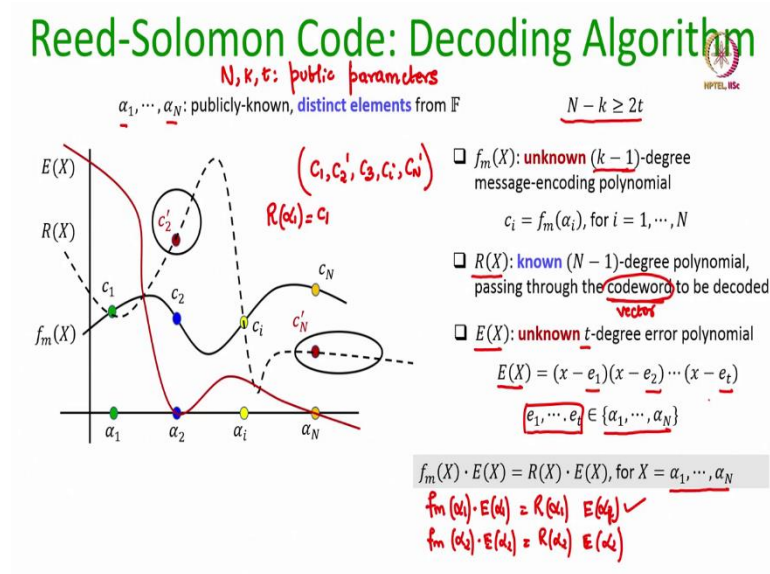
So, we can imagine that from the viewpoint of the receiver, it has actually a polynomial  $R(X)$  whose degree is  $N - 1$  passing through the codeword which the receiver wants to decode. I should not use the term codeword, I should rather use the term vector because the received vector need not correspond to a codeword. Namely, it knows a polynomial such that  $R(\alpha_1) = c_1, R(\alpha_2) = c_2', R(\alpha_i) = c_i, R(\alpha_N) = c'_N$ .

So, the circled components here are the ones which have been corrupted. From the viewpoint of the receiver there is also an unknown error polynomial whose degree is  $t$  and whose roots are the locations at which errors have been introduced. So, remember there are up to  $t$  locations where error could have been introduced. So, let us denote those unknown error locations or the indices as  $e_1, e_2, \dots, e_t$  and they could be any  $t$  locations, any  $t$  indices from this collection.

So, for instance in this example, the errors have occurred at location 2 and location  $N$ . So,  $e_1$  is basically  $\alpha_2$  and  $e_2$  is  $\alpha_N$ . So, these  $e$  components correspond to those alpha components, where errors have been introduced. So, there could be  $t$  such error components  $e_1, e_2, \dots, e_t$ . Again, they are not known yet to the receiver.



(Refer Slide Time: 20:38)

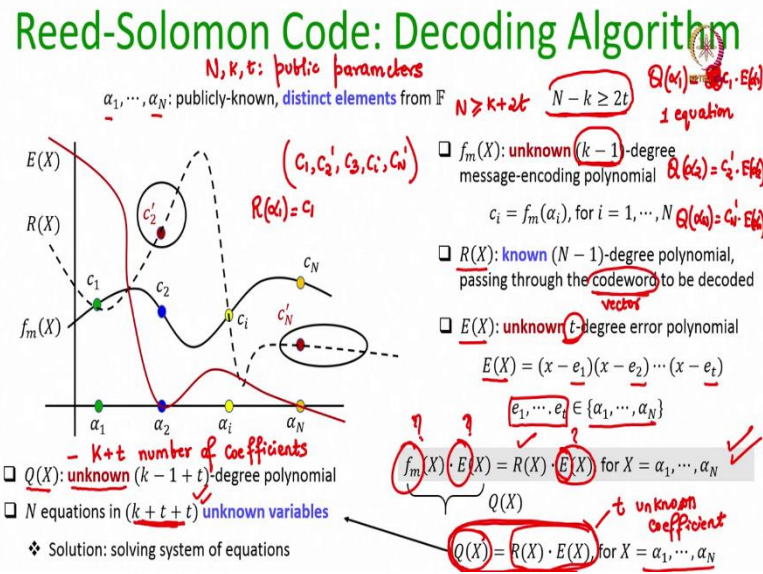


But it knows the relationship that whatever are those unknown variables, they are from the set  $\{\alpha_1, \dots, \alpha_N\}$ . Now, we know that for the receiver this equation holds. Namely, the product of the unknown message encoding polynomial with the error polynomial is the same as the product of the received polynomial and the error polynomial at all the  $X$  values ranging from  $\alpha_1, \dots, \alpha_N$ .

Say for instance, let us take it for those  $\alpha$  indices where errors have not occurred. So, at  $\alpha_1$  the error has not occurred; that means,  $f_m(\alpha_1)$  will be  $c_1$  and  $R(\alpha_1)$  is also  $c_1$ . So, that is same, both the sides are multiplied with  $E(\alpha_1)$  and  $E(\alpha_1)$  is anyhow same. So, this holds whereas, at  $E(\alpha_2)$  where error has occurred in this specific example, the value of  $f_m$  at  $\alpha_2$  is  $c_2$ .

And the value of  $R(\alpha_2)$  is  $c'_2$  which are different that is fine, but as soon as I multiplied both the sides with  $E(\alpha_2)$  the product becomes 0. Because my definition of  $E(X)$  polynomial is that it vanishes at all those  $\alpha$  components corresponding to which errors have been introduced and this relationship basically constitutes the crux of the decoding algorithm.

(Refer Slide Time: 22:34)



So, even though receiver does not know what is the  $f_m$  polynomial, what the  $E$  polynomial is. It only knows the  $R$  polynomial, it knows that this relationship holds. Now, let us denote the product polynomial on the left-hand side of this equation by the  $Q(X)$  polynomial.

So, we can rewrite the equation as  $Q(X)$  will be same as the product of the  $R$  polynomial and the error polynomial for  $X = \alpha_1$  to  $\alpha_N$ . Now, if we focus on this equation, what is the degree of the  $Q(X)$  polynomial? It is a  $k-1+t$  degree polynomial because,  $f_m$  polynomial has a degree  $k-1$ . The  $E$  polynomial can have degree  $t$  and since  $Q$  is the product of two polynomials over a field, its degree will be bounded by the sum of the two polynomials.

So, one of the polynomials have degree  $k-1$ , another polynomial has degree  $t$ . So, that is why the overall degree of  $Q(X)$  will be  $k-1+t$ . Namely, it will have  $k+t$  number of coefficients. Now, are those coefficients known to the receiver? The answer is no, they are unknown, all of them, because none of the two individual polynomials is known to the receiver. Now, what about. So, that is one fact.

And, now since receiver knows the value, the receiver knows that this equation holds for  $N$  number of  $X$  values. If we imagine that  $Q(X)$  is an unknown polynomial with  $k+t$  number of coefficients and  $E(X)$  is another polynomial with  $t$  unknown coefficients, then overall for the receiver there are  $k+2t$  number of unknown coefficients.  $k+t$  number of coefficients are unknown from the  $Q(X)$  polynomial and  $t$  number of coefficients are

unknown due to the  $E(X)$  polynomial. Why  $t$  coefficients from the  $E(X)$  polynomial? Because  $E(X)$  is a  $t$  degree polynomial.

So, overall, for the receiver, there are total  $k + 2t$  number of unknown variables. But the receiver also now has  $N$  number of equations in those unknown variables right. So, we can interpret  $Q(X)$  as an unknown polynomial, where there are  $k + t$  unknowns and the polynomial on the RHS will be a polynomial where some part is known, and some part is unknown.

So, by rearranging the polynomial terms in the polynomial, the polynomial on the right-hand side will be a polynomial where there are  $t$  number of unknowns. So, total for the receiver there are  $k + 2t$  number of unknowns. And now if receiver substitutes  $Q(\alpha_1) = R(\alpha_1)$  and  $R(\alpha_1)$  is basically  $c_1$  which it has received followed by  $E(\alpha_1)$ , that gives him one equation in those  $k + 2t$  variables.

Similarly, if it substitutes  $Q(\alpha_2) = c'_2 \times E(\alpha_2)$ , that gives him another equation in the same  $k + 2t$  unknown variables. And, like that once it substitutes  $Q(\alpha_N) = c'_N \times E(\alpha_N)$  that gives him another equation in the same  $k + 2t$  unknown variables. So, now, there are  $k + 2t$  number of unknown variables and  $N$  number of equations.

And remember that we are working with this condition,  $N \geq k + 2t$ . So, it has at least as many equations as the number of unknown variables. And, by solving that system of equations, it can find out those  $k + 2t$  unknown variables. And, once it finds out those  $k + 2t$  unknown variables, it will know the error location and that helps him to find out the exact identity of the locations at which error has occurred.

And now, once it knows the exact identity of the locations where error has occurred, it can find out the message encoding polynomial by ignoring those components. By ignoring those components, it will be now able to find out the original message encoding polynomial. From the message encoding polynomial receiver will be now able to identify the sender's actual message. So, this is how the decoding algorithm will work.

Thank you.